دليل الإحتراف

مع

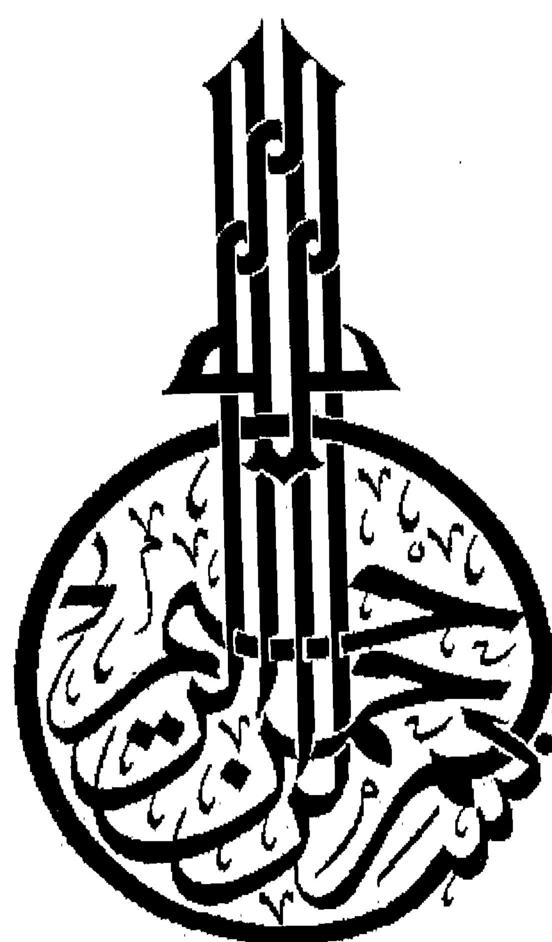
Visual Basic .net

فى تطبيقات الويندوز - قواعد البيانات
اللغة العربية - التقارير

رقم الإيداع بدار الكتب : ٢٠٠٣/٤٣٤١
الترقيم الدولي : ٤-٣٠٤-٢٨٧-٩٧٧

دار الكتب العلمية للنشر والتوزيع
٥٠ شارع الشيخ ريحان - عابدين - القاهرة
☎ ٧٩٥٤٢٢٩

لمزيد من المعلومات يرجى زيارة موقعنا على الأنترنت
www.scientificbookhouse.com
e-mail: sbh@link.net


بسم الله الرحمن الرحيم

مقدمة

مع مقدم Visual Basic .NET أصبحت لغة Visual Basic من بين لغات البرمجة التى تقوم على استخدام الكائنات (Object-Oriented Programming). ويعتبر إصدار Visual Basic.NET مختلف جدا عن الإصدارات السابقة من Visual Basic. ومع أن هذا الإصدار يحتوى على الكثير من ابتكارات البرمجة، إلا أن أهم هذه الابتكارات على الإطلاق هي التدعيم الكامل للبرمجة باستخدام الكائنات (OOP).

ولقد تم تصميم لغة Visual Basic .NET لكي يمكن استخدامها في إعداد تطبيقات الإنترنت المختلفة بالإضافة إلى التطبيقات الخاصة بنظام تشغيل الويندوز. ونظرا لتشعب أنواع التطبيقات التى يمكن تنفيذها باستخدام هذه اللغة، مما يتطلب كتابة العديد من الكتب لتفصيلها، لذلك قررت التركيز على جانب واحد هو جانب التطبيقات العاملة تحت نظام الويندوز. ويرجع ذلك إلى أن معظم التطبيقات العاملة الآن تستخدم هذا النظام، كما أن العناصر المستخدمة فى إعداد تطبيقات الويندوز لاتختلف عن تلك المستخدمة فى غيرها من التطبيقات.

وكان الهدف الأول أثناء كتابة هذا الكتاب هو التركيز على عناصر البرمجة الجوهرية التى يحتاج إليها المبرمج. ولذلك تم تقديم موضوعات استخدام Visual Basic .NET فى تكوين تطبيقات الويندوز باللغة العربية. كما تم التركيز على موضوعات إعداد التقارير باستخدام مولد التقارير Crystal Reports المعتمد لدى Visual Basic. وكان التركيز الأكبر على استخدام قواعد البيانات لأن التطبيقات التى تستخدم قواعد البيانات تمثل النسبة الأكبر من بين جميع التطبيقات. ولقد تم تقسيم الكتاب على أساس التدرج فى عرض الموضوعات التى تتكون منها التطبيقات بحيث تكون معرفة أحد الموضوعات ضرورية قبل الوصول إلى الموضوع الذى يليه. على هذا الأساس تم تقسيم الكتاب إلى ستة فصول على الترتيب التالى:

الفصل الأول: يتناول الأساسيات التى تقوم عليها لغة Visual Basic .NET. هذه الأساسيات تشمل ثلاثة عناصر أساسية: نظام NET Framework، بيئة Visual Studio .NET، وعناصر البرمجة باستخدام الكائنات.

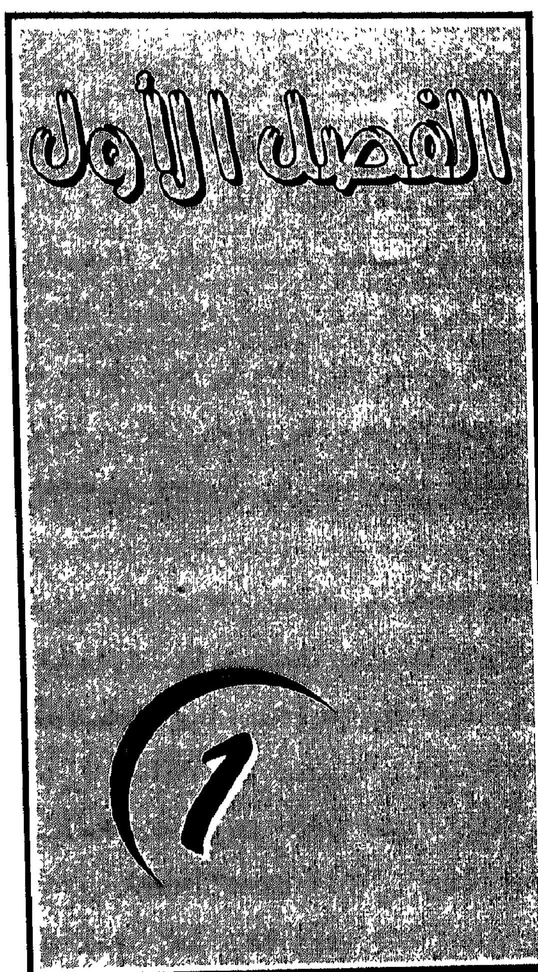
الفصل الثانى يتناول قواعد لغة Visual Basic المستخدمة فى كتابة الكود الذى يمكن تنفيذه بواسطة الكمبيوتر. تتعلق هذه القواعد بعناصر بناء البرامج التى تشمل مواقع تخزين البيانات، أنواع البيانات التى يتم تخزينها، العوامل التى تقوم بتنفيذ العمليات على البيانات، والتعليمات الخاصة بالتنفيذ.

الفصل الثالث يركز على استخدام نماذج الويندوز. وتم تقسيم هذا الفصل إلى جزأين هما نماذج الويندوز، واستخدام اللغة العربية فى تطبيقات نماذج الويندوز.

الفصل الرابع يشتمل على تفصيلات أدوات التحكم والمكونات المستخدمة فى بناء واجهات تطبيقات نماذج الويندوز. وينقسم إلى قسمين، الأول خاص بأدوات التحكم المرئية والثانى يختص بأدوات التحكم غير المرئية.

الفصل الخامس يشتمل على معالجة الموضوعات الخاصة بالتعامل مع قواعد البيانات فى Visual Basic .NET. ويقدم هذا الفصل تفصيلات التعامل مع تكنولوجيا ADO.NET الجديدة بالإضافة إلى كيفية التعامل مع كائنات فئات البيانات (datasets) التى تم تقديمها ضمن هذه التكنولوجيا.

الفصل السادس يتناول موضوعات إعداد التقارير باستخدام Crystal Reports فى بيئة ADO.NET الجديدة. ولقد تم التركيز بصفة خاصة على موضوع إعداد التقارير باستخدام فئات البيانات.



الركائز التى تقوم عليها تطبيقات Visual Basic.NET

يمكن تعريف لغة Visual Basic .NET بأنها الجيل الجديد فى هذه اللغة الذي تقدمة شركة مايكروسوفت لإستخدامه فى تطوير معظم أنواع التطبيقات. الركيزة الأولى التي يقوم عليها تطوير التطبيقات باستخدام هذه اللغة هى العمل داخل النظام الذى يطلق علية إطار الشبكة (NET Framework.). هذا النظام قدمته مايكروسوفت لتبسيط تطوير التطبيقات المختلفة، وخصوصا تطبيقات الوب. على هذا الأساس، لا يمكن تكوين تطبيقات Visual Basic .NET إلا بعد تثبيت نظام NET Framework. ويعتبر هذا النظام فى نفس الوقت هو الركيزة الأساسية التى تبنى عليها باقى مكونات بيئة التطوير، التى تشمل الاستديو المرئى (Visual Studio .NET) وتصنيفات الأنواع المختلفة المستخدمة فى البرمجة باستخدام الكائنات (Object-Oriented Programming).

الركيزة الثانية لتطوير تطبيقات Visual Basic .NET هى استخدام الاستديو المرئى (Visual Studio .NET)، الذى يمثل منصة عمل لتكوين التطبيقات باستخدام الوسائل المرئية. ويؤدى استخدام Visual Studio .NET إلى إضفاء الآلية على إجراءات تكوين التطبيقات. على سبيل المثال، من الصعوبة تكوين واجهات الاستخدام الرسومية مثل القوائم، شرائط الأدوات، وأدوات التحكم الأخرى باستخدام الكود، بينما يمكننا تكوين هذه العناصر بسهولة عن طريق السحب والإسقاط (drag and drop) فى Visual Studio .NET. بل إن Visual Studio .NET يمكننا من تكوين تطبيق ويندوز كامل بدون كتابة سطر واحد من الكود. ومع أن مثل هذا التطبيق يمثل الصورة الأولية للتطبيقات، لأنة لا يشتمل إلا على نافذة تحتوى على الوظائف الأولية التى يوفرها الويندوز، إلا أنة يمثل نقطة البداية فى إنشاء التطبيقات المختلفة.

وتمثل البرمجة باستخدام الكائنات (Object-Oriented Programming) الركيزة الثالثة اللازمة لإنشاء تطبيقات Visual Basic.NET. ويمكن تعريف الكائنات بأنها مرحلة متطورة فى وصف الموضوعات التى تتكون منها البرامج. لقد كانت موضوعات البرامج فى السابق يتم وصفها باستخدام بند واحد من المعلومات، وهو ما لا يتفق مع احتواء هذه الموضوعات على عناصر معلومات متعددة ومتشابكة. وللتخلص من هذه المشكلة، اتجهت لغات البرمجة الحديثة إلى معاملة الأشخاص، الأشياء، والموضوعات التى يستخدمها البرنامج على أنها

كائنات. من أمثلة هذه الكائنات النماذج التى نستخدمها فى تطبيقات الويندوز، التى تعتبر تمثيل مبرمج للنافذة. ويمكن أن يحتوى الكائن على كائنات أخرى. على سبيل المثال، كائن السيارة يحتوى على بنود متعددة من المعدات، مثل المحرك، الإطارات، جهاز تحويل الحركة وخزان الوقود، التى يعتبر كل منها كائن فى حد ذاته. قياسا على ذلك، نجد أن كائن النموذج فى Visual Basic يمكن أن يحتوى على أدوات تحكم (Controls) يعتبر كل منها كائن فى حد ذاته. من أمثلة الكائنات الموجودة على نموذج الويندوز، القائمة الرئيسية (Main Menu)، شريط الأدوات (Toolbar)، مربعات الحوار (Dialog boxes)، وغيرها من الكائنات التى توضع على سطح النموذج فى تطبيقات الويندوز. لكل كائن من هذه الكائنات مواصفاته الخاصة التى يتم التعبير عنها باستخدام مصطلحات، مثل الخصائص (Properties)، الوسائل (Methods)، والأحداث (Events). وبجانب المفاهيم السابقة، تشتمل البرمجة باستخدام الكائنات على مفاهيم أخرى، مثل الوراثة (Inheritance)، التغليف (Encapsulation)، و تعدد أشكال الكائنات (Polymorphism).

نظام تطبيقات الشبكة

يمكن تكوين تطبيق Visual Basic .NET بدون استخدام Visual Studio .NET، عن طريق استخدام محرر نصوص مثل برنامج Notepad وسطر الأوامر . ولكن لايمكن تكوين هذه التطبيقات، سواء باستخدام محرر النصوص أو Visual Studio، إلا بعد تثبيت نظام تطبيقات الشبكة (NET Framework). بإيجاز لايمكن تطوير تطبيقات NET بدون تثبيت نظام NET Framework. ويقصد بكلمة Framework البنية التى تدعم باقى أجزاء البناء.

ويمكن النظر إلى نظام NET Framework على أنة إطار عمل جديدة يهدف إلى تبسيط تطوير تطبيقات الكمبيوتر فى بيئة الإنترنت التى تعتمد على التوزيع الواسع النطاق للتطبيقات. ولقد تم تصميم هذا النظام لتحقيق الأهداف التالية :

- توفير بيئة ثابتة للبرمجة باستخدام الكائنات (Object-Oriented Programming)، سواء تم تخزين الكود وتنفيذه محليا، تنفيذ الكود الموزع عبر شبكة الإنترنت محليا، أو تنفيذ الكود من بعيد.

- توفير بيئة مناسبة لتنفيذ البرامج تعمل على تقليص كتابة الكود إلى أقصى حد.
- توفير بيئة تضمن التنفيذ الأمن للبرامج، بما فى ذلك الكود الذى يجرى كتابته بواسطة أطراف أخرى غير معروفة أو غير موثوق بها بالكامل.
- توفير بيئة تنفيذية للبرامج تعمل على التخلص من مشاكل الأداء المرتبطة ببيئة البرامج غير المترجمة، التى تستخدم بيئة تفسير الكود (Interpreted environment).
- المحافظة على خبرة المبرمج الخاصة بأنواع التطبيقات المختلفة، مثل التطبيقات التى تعتمد على الويندوز والتطبيقات التى تعتمد على الوب.
- بناء جميع الاتصالات على معايير قياسية للتأكد من أن الكود الذى يعتمد على NET Framework يمكن أن يتكامل مع أى كود أخر.

ويتكون نظام NET Framework من مكونين رئيسيين: مدير تشغيل الكود (Common Language Runtime) و مكتبة التصنيفات (NET Framework class library).

مدير تشغيل الكود

المكون الأول فى نظام Net Framework هو مدير تشغيل الكود (Common language runtime)، الذى يمثل البنية الأساسية لنظام NET Framework. يمكن النظر إلى هذا المكون على أنة وكيل يدير الكود فى وقت التشغيل ويقوم بتوريد الخدمات المركزية التى تجرى الحاجة إليها، مثل إدارة الذاكرة (memory management)، إدارة سلاسل العمليات (thread management)، الاتصال من بعد (remoting)، وفى نفس الوقت فرض الاستخدام الأمن لأنواع البيانات المختلفة، وغيرها من أشكال المحافظة على دقة الكود. ويطلق على الكود الذى يلتزم بهذا النظام تعبير الكود الذى يتم إدارته (managed code)، كما يطلق على الكود الذى لا يلتزم به تعبير الكود الذى لا تتم إدارته (unmanaged code).

ويمكننا حصر الوظائف الأساسية التى يقوم بها مدير تشغيل الكود فيما يلى :

١. فرض سرية الوصول إلى الكود. على سبيل المثال، لا تستطيع البرامج العاملة على صفحات الوب الوصول إلى بيانات المستخدم الشخصية، نظام الملفات، أو الشبكة التى يعمل عليها.

٢. فرض تدقيق الكود المستخدم عن طريق التحقق من الأنواع والكود باستخدام بنية أساسية يطلق عليها Common Type System(CTS).

٣. التخلص من خطأين شائعين بالتطبيقات هما: تسرب الذاكرة (Memory Leaks) ووجود المراجع غير الصالحة للاستخدام.

٤. يقوم مدير تشغيل الكود بتسريع إنتاجية المبرمج. على سبيل المثال، يمكن أن يقوم المبرمج بكتابة تطبيقاته باستخدام اللغة التى يفضلها، ولكنة فى نفس الوقت يمكنة الاستفادة الكاملة من مدير التشغيل، مكتبة التصنيفات، والمكونات المكتوبة بلغات أخرى وبواسطة مبرمجين آخرين. وللاستفادة من ذلك يجب أن يتم تصميم برامج ترجمة الكود (Compilers) على أساس استخدام نظام NET Framework.

٥. بالإضافة إلى أن نظام مدير التشغيل (Runtime) قد تم تصميمه لبرامج المستقبل، إلا أنة يدعم البرامج الحالية والبرامج السابقة. من الأمثلة على ذلك، توفر إمكانية استخدام مكونات COM وDLLs بواسطة المبرمجين عند الحاجة إليها.

٦. يحتوى مدير تشغيل الكود على خاصية يطلق عليها Just-in-time تقوم بترجمة الكود وقت الحاجة. تؤدى هذه الخاصية إلى تمكين الكود الذى يجرى إدارته من العمل بلغة الآلة التى يتم التنفيذ عليها.

٧. يمكن استضافة مدير التشغيل بواسطة التطبيقات العاملة على الخادم، مثل MS SQL Server، مما يمكننا من استخدام الكود الذى يمكن إدارته فى كتابة إجراءات النشاط.

مكتبة التصنيفات

المكون الثانى فى نظام NET Framework. هو مكتبة التصنيفات (class library). تحتوى هذه المكتبة على مجموعة شاملة من أنواع التصنيفات التى تتكامل بدقة مع مدير تشغيل الكود، يمكن إعادة استخدامها، وتوفر أساس البرمجة باستخدام الكائنات. ويمكن استخدام هذه المكتبة فى تطوير أنواع التطبيقات، التى تبدأ من تطبيقات سطر الأوامر التقليدية وتطبيقات الواجهات الرسومية إلى تطبيقات الوب.

وتمكننا الأنواع التى تحتوى عليها مكتبة التصنيفات من إنجاز مجموعة من مهام البرمجة الشائعة، مثل إدارة سلسلة، تكوين مجموعات البيانات، الاتصال مع قواعد البيانات، والوصول إلى الملفات. وبالإضافة إلى هذه المهام العامة، تشتمل مكتبة التصنيفات على أنواع تدعم مجموعة متنوعة من سيناريوهات التطوير، مثل تطوير تطبيقات الوحدات الطرفية (Console applications)، تطوير تطبيقات الوب (Web applications)، وتطوير تطبيقات الويندوز (Windows applications). على سبيل المثال، تتطلب تطبيقات الويندوز استخدام مجموعة من التصنيفات التى تقوم بتبسيط عمليات تطوير الواجهات الرسومية بتطبيقات الويندوز.

الاستديو المرئي

يمكن تعريف الاستديو المرئي (Visual Studio NET) بأنة مجموعة كاملة من أدوات التطوير التى تستخدم فى بناء أنواع متعددة من التطبيقات، مثل تطبيقات الوب ASP، خدمات الوب XML، وتطبيقات سطح المكتب المختلفة. وتشترك لغات Visual Basic.NET، Visual C++ .NET، و Visual C# NET فى استخدام بيئة التطوير المتكاملة (integrated development environment)، التى يوفرها Visual Studio.NET ويطلق عليها اختصارا (IDE). تسمح بيئة التطوير المذكورة لهذه اللغات بالمشاركة فى استخدام الأدوات والتسهيلات الموجودة فى Visual Studio NET، لتكوين حلول مكتوبة بلغات مختلطة.

إصدارات Visual Studio.NET

هناك أربعة إصدارات من Visual Studio.NET: إصدار المحترفين (Professional)، إصدار المبرمجين بالمنشات التجارية (Enterprise Developer)، إصدار المخططين بالمنشات التجارية (Enterprise Architect)، الإصدار الجامعي (Academic). كما تقدم اللغات العاملة تحت Visual Studio NET، أيضا إصداراتها الخاصة. يعرض الجدول رقم (١) قوائم بأهم الإمكانيات المتاحة تحت كل إصدار من إصدارات Visual Studio .NET السابق الإشارة إليها.

الإمكانيات المتاحة	Professional	Enterprise Developer	Enterprise Architect	Academic
استخدام Visual Basic .NET	X	X	X	X
استخدام Visual C# .NET	X	X	X	X
استخدام Visual C++ .NET	X	X	X	X
بناء واستخدام خدمات XML على الوب	X	X	X	X
بناء تطبيقات الوب	X	X	X	X
بناء تطبيقات الويندوز	X	X	X	X
تصميم الجداول والمشاهد على SQL Server الخاص بسطح المكتب	X	X	X	X
تصميم الجداول ، المشاهد ، الإجراءات ، لدوال ، وغيرها على SQL Server و Oracle		X	X	
إصدار المبرمج من Windows 2000 Sever		X	X	
إصدار المبرمج من SQL Server 2000		X	X	

جدول ١

العتاد المطلوب لتثبيت Visual Studio .NET

يتطلب تثبيت الاستديو المرئي (Visual Studio NET) توفر عتاد محدد قد يختلف باختلاف إصدارات Visual Studio NET. ولهذا يجب أن يتوفر فى الجهاز الذى نقوم بالتثبيت علية ، المتطلبات المعروضة بالجدول رقم (٢).

المتطلبات	إصدار Visual Studio .NET			
	Enterprise Architect	Enterprise Developer	Professional	Academic
المعالج Processor	كمبيوتر شخصي به معالج Pentium ، ٤٥٠ MHz. ويفضل Pentium111 ، ٦٠٠ MHz.	نفسة	نفسة	نفسة
الذاكرة RAM	٩٢ MB لويندوز ٢٠٠٠ إصدار المحترفين ؛ ١٩٢ MB لويندوز ٢٠٠٠ إصدار الخوادم ؛ ١٦٠ MB لويندوز XP	نفسة	نفسة	نفسة
القرص الصلب Hard Disk Space	٥٠٠ MB لتثبيت ملفات بنظام التشغيل ، ٣ GB لتثبيت باقى ملفات التطبيق.	نفسة	نفسة	نفسة
نظام التشغيل Operating System	ويندوز ٢٠٠٠ ، ويندوز XP ، أو ويندوز NT 4.0 .	نفسة	نفسة	نفسة
سواقة CD أو DVD	مطلوب	مطلوب	مطلوب	مطلوب
موفق Video	درجة حدة ٨٠٠x٦٠٠ – ٢٥٦ لون	نفسة	نفسة	نفسة
فارة Mouse	فارة مايكروسوفت أو متوافقة معها	نفسة	نفسة	نفسة

جدول ٢

ويمكننا تحسين أداء Visual Studio .NET باتباع الطرق التالية :

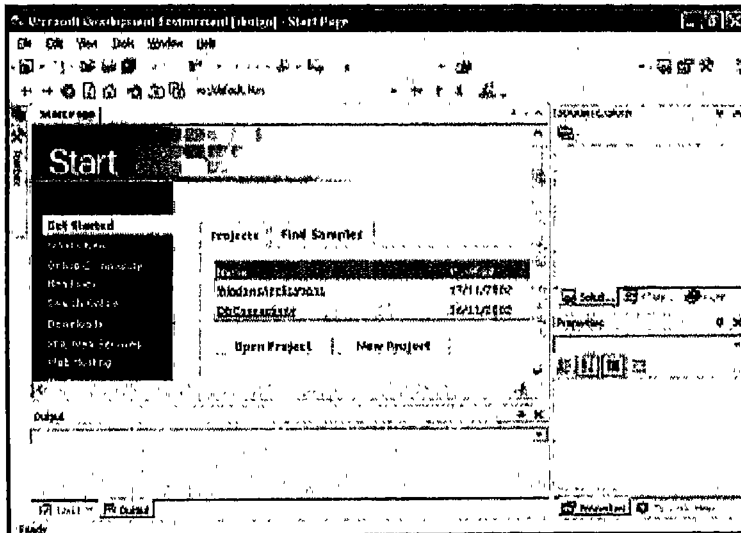
- إقفال تطبيقات البحث عن الفيروسات.
- تشغيل برنامج Defragment بعد تثبيت هذا المنتج للتخلص من تكسير الملفات على

القرص الصلب.

- التحقق من وجود المتطلبات المفضلة المتعلقة بالذاكرة والمعالج الخاصة بالإصدار الذى تم تثبيته. وترقية الذاكرة والمعالج عند الحاجة إلى ذلك.
- وعند فتح بيئة التطوير، يجب القيام بما يلي لتحسين الأداء:
- نقوم بإقفال نوافذ الأدوات التى لا نستعملها عند البدء، قبل إقفال Visual Studio .NET لزيادة سرعة بدء التطبيق عند فتحة فى المرة التالية.
- لا نختار عرض نافذة الخصائص عند بدء تشغيل بيئة تطوير التطبيقات (IDE). حيث يتم عرض هذه النافذة تلقائيا عند فتح أحد الحلول.
- لا نختار عرض صفحة البداية أو المساعدة الديناميكية عند بدء التشغيل لزيادة سرعة بدء التطبيق.

تهيئة استخدام Visual Studio.NET

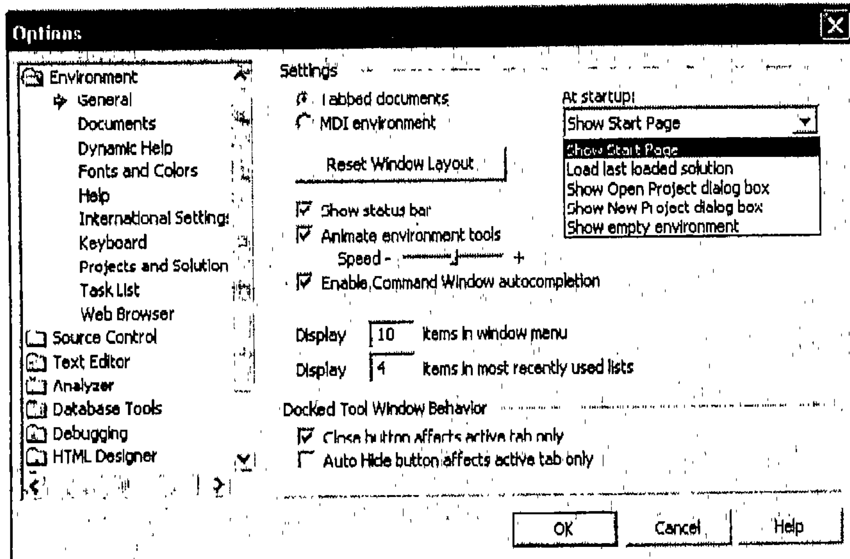
لبدء استخدام Visual Studio .NET بعد تثبيته، نقوم بالنقر بالترتيب على زر البداية (Start)، Programs، Microsoft Visual Studio.NET، Microsoft Visual Studio.NET. يترتب على ذلك عرض صفحة البداية، التى يبينها الشكل رقم (١).



شكل رقم ١

يوضح الشكل السابق أن صفحة البداية فى Visual Studio .NET تتكون من خمسة مربعات فرعية بجانب شرائط الأدوات والقائمة الرئيسية وشريط المعلومات. ويمكن استخدام المربعات الفرعية فى عرض عدد كبير من النوافذ. وعندما يتم عرض نافذة فى مربع من المربعات، يتم فى نفس الوقت إضافة ملصق باسم النافذة المضافة. فى منتصف الصفحة، يوجد مربع يستخدم لعرض نافذة التصميم و نافذة تحرير الكود بصفة رئيسية بجانب نوافذ أخرى. على الجانب الأيمن يوجد مربعان، الأعلى لعرض نافذة Solution Explorer ونوافذ أخرى، والأسفل لعرض نافذة Properties و نوافذ أخرى. فى المربع الموجود فى أقصى اليسار، يتم عرض نافذة ToolBox ونافذة Server Explorer ونوافذ أخرى. فى أسفل الصفحة، يوجد مربع يتم إستخدامة فى عرض عدد من النوافذ من أهمها نافذة Output ونافذة Task List.

ويمكن تغيير مظهر صفحة البداية فى Visual Studio.NET عن طريق اختيار Tools من القائمة الرئيسية ثم Options. يترتب على ذلك عرض مربع حوار Options المبين بالشكل رقم (٢). فى مربع الحوار المذكور ننقر على السهم المتجه إلى الأسفل فى مربع السرد المركب تحت عنوان At startup.



شكل رقم ٢

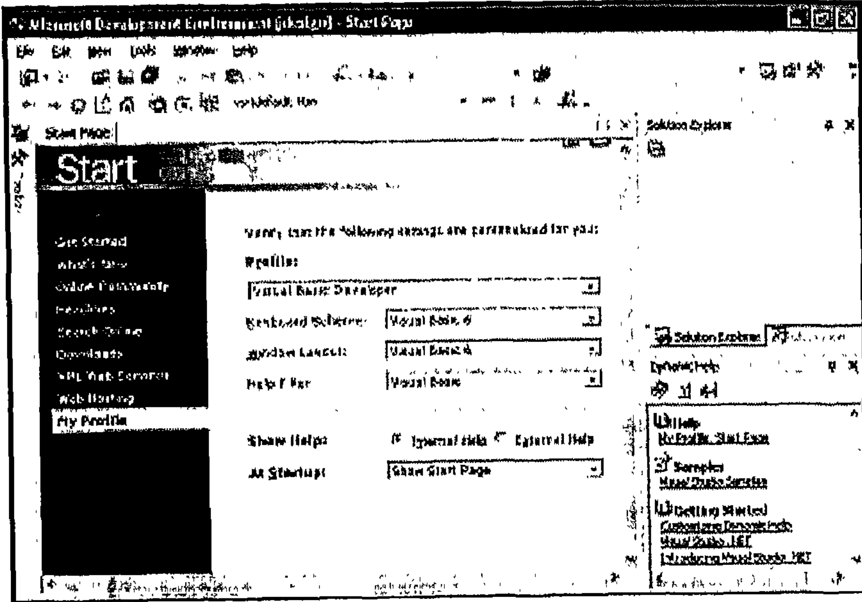
يعرض مربع الحوار رقم (٢) الخيارات التالية:

- Show Start page. يعرض هذا الاختيار الشكل الافتراضي المبين فى الشكل رقم (١).
- Load last loaded solution. يترتب على هذا الخيار قيام Visual Studio.NET بعرض أخر المشروعات السابق العمل معها، كما ظهر عند إقفال Visual Studio.NET.
- Show Open Project dialog box. يعرض هذا الخيار مربع حوار Open Project، الذى يمكن عرضة أيضا باختيار File من القائمة الرئيسية ثم اختيار Open، Projectعلى الترتيب.
- Show New Project dialog box. يترتب على هذا الخيار عرض مربع حوار New Project، الذى يمكن عرضة باختيار File من القائمة الرئيسية ثم New، Project على الترتيب.
- Show empty environment. يعرض هذا الاختيار نموذجا خاليا من بيئة التطوير المتكاملة (IDE).

ويشترك عدد من لغات الكمبيوتر فى استخدام بيئة التطوير التى يوفرها Visual Studio.NET. تشمل هذه اللغات بجانب Visual Basic.NET، لغات أخرى مثل Visual C++ و Visual C#. ومع أن ذلك يعتبر تطور إيجابي إلا أن له جانبه السلبي. على سبيل المثال، عند استخدام المساعدة الداخلية لا تستطيع بيئة التطوير المتكاملة (IDE) معرفة ما إذا كان المستخدم يطلب المساعدة التى تخص Visual Basic أو غيره من اللغات المشتركة معه فى بيئة التطوير.

ويمكن تزويد Visual Studio.NET بمعلومات عن المستخدم لتمكينه من تعديل بيئة التطوير (IDE) بما يتناسب مع اختيارات هذا المستخدم. ويمثل هذا التعديل القيم الافتراضية الخاصة بمظهر النوافذ، اختصارات لوحة المفاتيح، المساعدة، وغيرها من موضوعات التهيئة المختلفة. ويمكن تزويد Visual Studio.NET بمعلومات عن المستخدم عن طريق اختيار MyProfile فى صفحة البداية الموضحة بالشكل رقم (١). يترتب على هذا الاختيار عرض صفحة

My Profile المبينة بالشكل رقم (٣). من قائمة Profile المنسدلة، نختار Visual Basic Developer. بعد إدخال هذا الاختيار، يقوم Visual Studio.NET بمعالجة التغييرات المرتبطة به، مثل تغيير مخطط لوحة المفاتيح، وتغيير خريطة عرض النوافذ ثم يعيد عرض الصفحة الافتتاحية.



شكل رقم ٣

تكوين تطبيقات الويندوز باستخدام Visual Studio .NET

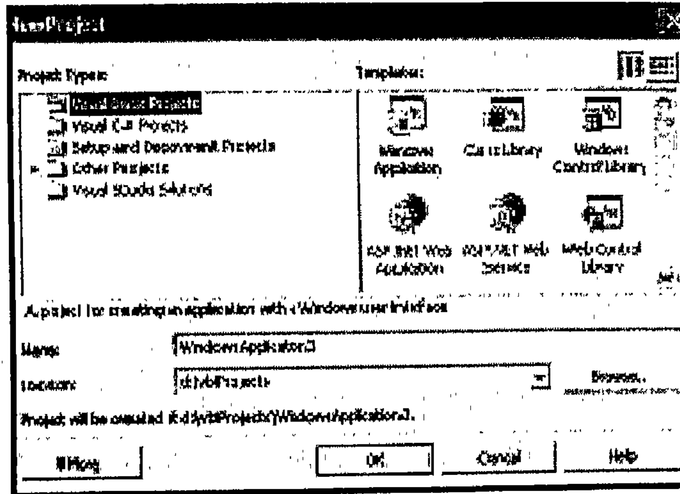
يمكننا Visual Studio .NET من تكوين أنواع كثيرة من التطبيقات، تشمل تطبيقات نماذج الويندوز (Windows Forms Applications)، تطبيقات نماذج الوب (Web Forms Applications)، تطبيقات الوحدات الطرفية (Console). نماذج الويندوز التى يطلق عليها غالبا WinForms، تكون واجهة الاستخدام فى تطبيقات الويندوز التى يجرى تشغيلها دائما على كمبيوتر محلى، كما يمكن تشغيلها على خادم فى شبكة. وتقدم لنا نماذج الوب واجهة الاستخدام الخاصة بتطبيقات تصفح الوب على شبكة الإنترنت. من أمثلة هذه التطبيقات موقع شركة طيران على شبكة الإنترنت يستخدم فى إدخال معلومات السفر والحصول على جدول الرحلات المتاحة. وتطبيقات الوحدات الطرفية هى من التطبيقات

التى لم تسمح بها إصدارات Visual Basic السابقة، غير أنها أصبحت متاحة فى Visual Basic .NET. ولا تستعمل تطبيقات الوحدات الطرفية الواجهات الرسومية التى تستعملها تطبيقات الويندوز، ولكنها تستعمل النصوص فى تكوين واجهات الاستخدام. والاختلاف بين التطبيقات الرسومية وبين تطبيقات واجهات النصوص أعمق كثيرا من الاختلاف بين واجهات الاستخدام لكل منهما.

لإيضاح كيفية استخدام بيئة تطوير Visual Studio .NET فى إنشاء تطبيق ويندوز يستخدم لغة Visual Basic.NET، نقوم بتنفيذ الخطوات التالية:

١. نبدأ تشغيل Visual Studio .NET بالنقر على Start ثم اختيار Visual Studio .NET من قائمة البرامج لنحصل على صفحة البداية.

٢. فى صفحة البداية، نختار File من القائمة الرئيسية، نشير إلى New ثم ننقر Project. يترتب على ذلك عرض مربع حوار New Project المبين بالشكل رقم(٤).



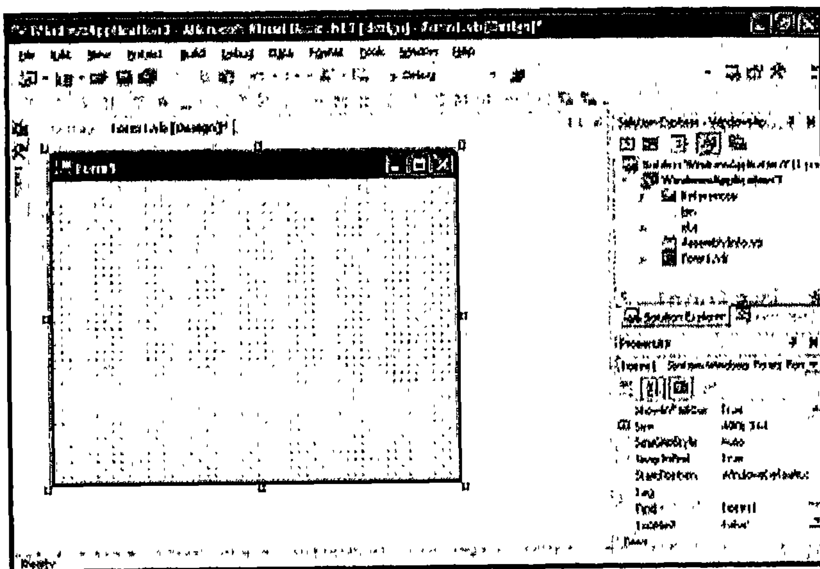
شكل رقم ٤

٣. فى الجانب الأيسر من مربع حوار New Project، نختار Visual Basic Project. وفى الجانب الأيمن نختار Windows Application.

٤. ندخل اسم للمشروع أو نترك الاسم الإفتراضى فى مربع Name.

٥. في مربع Location ندخل مسار حفظ المشروع، أو نترك المسار الإفتراضى كما هو. ويمكن تغيير المسار الإفتراضى للمشروعات التى نقوم بتكوينها عن طريق اختيار Tools ثم Options من القائمة الرئيسية لعرض مربع حوار Options. فى مربع حوار Options، نختار Project and Solutions من المربع الذى على يسار النافذة. ندخل المسار الإفتراضى فى مربع نص Visual Studio Projects location ثم ننقر OK.

٦. ننقر زر OK فى مربع New Project بعد الانتهاء من إدخال اسم وموقع المشروع. يترتب على ذلك، قيام Visual Studio .NET بتوليد الملفات والمجلدات التى يحتاجها المشروع ثم عرض النافذة المبينة بالشكل رقم (٥).



شكل رقم ٥

٧. يجب بعد ذلك بناء التطبيق باختيار Build, Build Solution من القائمة الرئيسية أو اختيار Build WindowsApplication1 من نفس القائمة. والفرق بين الاختيارين هو أن الأول يتعلق بكامل الحل (Solution) الذى قد يحتوى على أكثر من مشروع، بينما يتعلق الثانى بمشروع واحد فقط داخل الحل. ويحتوى المشروع الناتج على

جميع الملفات والروابط اللازمة للتطبيق.

٨. يمكن الآن تشغيل المشروع باستخدام Debug, Start من القائمة الرئيسية أو الضغط على مفتاح F5 بلوحة المفاتيح.

الكود الأساسي لتكوين تطبيق الويندوز

تم فى القسم السابق تكوين تطبيق Visual Basic.NET خاص بنماذج الويندوز عن طريق استخدام بيئة تطوير التطبيقات فى Visual Studio.NET. ذلك التطبيق هو عبارة عن نموذج واحد باسم Form1 يحتوى على الوظائف الأساسية التى تقوم بها نافذة تقليدية، مثل تحريك النافذة بالنقر على شريط العنوان (Title Bar) ثم سحب النموذج إلى موقع جديد، و تغيير حجم النموذج بوضع مؤشر الماوس على حدود النموذج وعند تحوله إلى سهمين متقابلين بالرأس نقوم بنقر وسحب حافة النموذج.

ومن الملاحظ فى التطبيق المذكور، أننا لم نكتب أى سطر من الكود، بل قام Visual Studio.NET بكتابة الكود الضروري لتكوين هذا التطبيق ومنحة الوظائف التى يقوم بها. ويمكن رؤية هذا الكود عن طريق اختيار View, Code من القائمة الرئيسية لكى نحصل على الكود التالى :

```
-Public Class Form1
    Inherits System.Windows.Forms.Form
+Windows Form Designer generated code

End class
```

تشير علامة + إلى أن إخفاء أحد الأقسام بالكود السابق. ولا يجب تغيير هذا الكود الذى قام بتكوينه Visual Studio.NET إلا بعد الفهم الكامل لهذا الكود حتى لا يتسبب ذلك فى فشل التطبيق. لفهم هذا الكود الذى قام Visual Studio.NET بانتاجة، ننقر على علامة + بجانب Windows Form Designer Generated Code لإظهار الكود التالى:

```
Public Class Form1
    Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "
```

```
        Public Sub New ()
            MyBase.New ()

            'This call is required by the Windows Form Designer.
            InitializeComponent( )

            'Add any initialization after the InitializeComponent () call

        End Sub

        'Form overrides dispose to clean up the component list.
        Protected Overloads Overrides Sub Dispose (ByVal disposing As Boolean)
            If disposing Then
                If Not (components Is Nothing) Then
                    components.Dispose ()
                End If
            End If
            MyBase.Dispose (disposing)
        End Sub

        'Required by the Windows Form Designer
        Private components As System.ComponentModel.IContainer

        'NOTE: The following procedure is required by the Windows Form Designer
        'It can be modified using the Windows Form Designer.
        'Do not modify it using the code editor.
        <System.Diagnostics.DebuggerStepThrough()>          Private          Sub
InitializeComponent()
            components = New System.ComponentModel.Container ()
            Me.Text = "Form1"
        End Sub

    #End Region

    End Class
```

لقد تم بناء الكود السابق على أساس استخدام الكائنات بدءا من تكوينها إلى تلاشيها. ولفهم هذا الكود نعرض فيما يلي شرحا للأجزاء المختلفة التى يتكون منها.

عبارة التصنيف (Class Statement)

تتكون عبارة التصنيف من ثلاثة أجزاء. الجزء الأول هو الكلمة المفتاحية Public التى تشير إلى أن التطبيقات الأخرى يمكن أن تستخدم هذا التصنيف. الجزء الثانى هو كلمة Class، التى يمكن ترجمتها إلى تصنيف. والتصنيف يمكن تعريفة بأنة قالب يحتوى على مجموعة من السمات والسلوكيات. ويستخدم هذا القالب فى تكوين الكائنات (Objects) التى تحمل نفس السمات ولها نفس السلوكيات الموجودة فى القالب المشتقة منة. من أمثلة الكائنات، النموذج (Form) الذى يتم عرضة عند تشغيل البرنامج السابق ويمثل واجهة الاستخدام. وكل كائن يجب اشتقاقه من أحد التصنيفات. وبالنظر إلى أنة يكون لدينا فى الغالب العديد من كائنات النماذج فى التطبيق الواحد ، لذا يجب التفرقة بين هذه النماذج عن طريق تمييز كل كائن باسم خاص به، هو الجزء الثالث الذى يأتى بعد كلمة Class فى الكود السابق. لقد قام Visual Studio.NET بتخصيص اسم Form1 للنموذج الإفتراضى الذى يحتوى علية التطبيق الذى يمثله الكود الموضح أعلاه. غير أنة يمكننا تغيير هذا الاسم، بل يجب القيام بذلك إذا كان التطبيق يحتوى على أكثر من نموذج.

عبارة الوراثة (Inherits Statement)

العبارة الثانية فى الكود السابق تتكون من جزأين. الجزء الأول هو كلمة يرث (Inherits)، التى تشير إلى أن تصنيفForm1 يرث جميع خصائص وسلوكيات التصنيف الذى يمثله الجزء الثانى من العبارة. الجزء الثانى يشمل اسم التصنيف الأساسى الذى يتم وراثة جميع تصنيفات النماذج منة، وهو System.Windows.Forms.Form. يوجد هذا التصنيف فى نطاق الأسماء System.Windows.Forms. ويمكن تعريف نطاق الأسماء بأنة مجموعة من التصنيفات ذات العلاقات فى مكتبة تصنيفات نظام NET Framework. وتعنى وراثة تصنيف Form1 من تصنيف Form، أن تصنيف Form1 به جميع خصائص النموذج القياسى، مثل شريط العنوان (Title Bar)، الحدود (Border)، ومنطقة العميل (Client Area). ويحتوى Form1 أيضا على جميع سلوكيات النافذة القياسية، مثل تحريك النموذج بالنقر على شريط عنوانه ثم سحبة، تغيير حجم النموذج بوضع مؤشر الماوس على حدود النموذج ثم سحبها إلى الخارج أو الداخل بعد تحول المؤشر إلى سهمين متقابلين بالرأس.

وسيلة New

التصنيفات ليست كائنات فى حد ذاتها ولكنها قوالب لإنتاج الكائنات التى يجرى استخدامها فى التطبيقات. وبالنظر إلى أن تطبيق نماذج الويندوز يتطلب وجود كائن نافذة (Window)، لذا يجب تكوين مثل من تصنيف Form يمثل كائن النموذج. ولكى يتم تكوين كائن النموذج، يجب استخدام الكود للقيام بهذه المهمة. ويطلق على الكود الذى يقوم بتكوين كائن من أحد التصنيفات إجراء بناء (Constructor). فى Visual Basic.NET، يوضع إجراء بناء الكائنات دائما فى وسيلة يطلق عليها NEW. على هذا الأساس، يقوم Visual Studio.NET بإنتاج وسيلة New التالية:

```
Public Sub New ()
  MyBase.New ()

  'This call is required by the Windows Form Designer.
  InitializeComponent ()

 'Add any initialization after the InitializeComponent () call

End Sub
```

يتم تنفيذ الكود الموجود فى وسيلة New عند استنساخ أحد الكائنات من التصنيف. الخطوة الأولى هى استدعاء وسيلة New بالتصنيف المستخدم فى بناء الكائن، وهو فى هذه الحالة تصنيف Form1. وتشير الكلمة المفتاحية MyBase إلى التصنيف الأصلى، وهو التصنيف الذى تحدده عبارة Inherits السابق إيضاحها. تقوم وسيلة New بعد ذلك باستدعاء وسيلة InitializeComponent. من الملاحظ أيضا فى الوسيلة السابقة، وجود سطرين يبدأن بعلامة اقتباس مفردة ('). يطلق على سطر الكود الذى يبدأ بعلامة الاقتباس المنفردة ملاحظة (Comment). ولا تعتبر سطور الملاحظات من كود التطبيق، ولكنها تستخدم فقط لشرح هذا الكود. وعند ترجمة التطبيق إلى كود تنفيذي، يقوم برنامج الترجمة بإهمال سطور الملاحظات.

وسيلة InitializeComponent

تقوم هذه الوسيلة بتخصيص قيم لخصائص متنوعة بالنموذج. والخاصية هى صفة من

صفات الكائن ويجب تحديد قيمة لها. على سبيل المثال، يحتوى كائن النموذج (Form) على خاصية BackColor التى تمثل لون خلفية النموذج. وما يحدد لون خلفية النموذج هو القيمة التى يتم تحديدها لهذة الخاصية. وإذا لم يتم تحديد قيمة للخاصية، يستخدم التطبيق القيمة الافتراضية التى يحددها Visual Studio.NET لتلك الخاصية عند غياب التوجيه من جانب المبرمج. على سبيل المثال، القيمة الافتراضية لخاصية BackColor تساوى gray. ويمكننا تغيير لون خلفية النموذج إلى أحد الألوان المتاحة عن طريق تخصيص القيمة الخاصة بها لتلك الخاصية.

ويطلق على عملية تحديد قيمة للخاصية مصطلح تخصيص (assignment). ويتم تنفيذ عملية التخصيص باستخدام عامل تخصيص يماثل علامة يساوى (=). وهناك العديد من عمليات التخصيص التى تجرى باستخدام الكود الذى تحتوى علية وسيلة InitializeComponent، كما يتضح من الكود التالى :

```
Private Sub InitializeComponent( )
    Me.AutoScaleBaseSize = New System.Drawing.Size (5, 13)
    Me.ClientSize = New System.Drawing.Size (292, 273)
    Me. Name = "Form1"
    Me. Text = "Form1"
End Sub
```

تشير الكلمة المفتاحية Me فى هذا الكود إلى مثل (instance) التصنيف الذى يتم فيه تنفيذ ذلك الكود. وهو فى هذه الحالة يشير إلى الكائن Form1، الذى يعتبر مثل من تصنيف النموذج. وتمثل الكلمة المفتاحية التى تأتى بعد كلمة Me اسم الخاصية. على سبيل المثال ، Me. Text تعنى خاصية Text فى كائن المثل الذى يجرى تنفيذه من التصنيف، وهو فى هذه الحالة كائن Form1. ويفصل بين المرجع إلى الكائن وبين اسم الخاصية بنقطة. وبعد اسم الكائن واسم الخاصية يأتى عامل التخصيص ثم القيمة التى يجرى تحديدها للخاصية. على سبيل المثال، يفيد كود "Me.Text = "Form1، أن خاصية Text فى كائن Form1 تساوى سلسلة "Form1".

وسيلة Dispose

تقوم هذه الوسيلة بعكس ما تقوم به وسيلة New. وسيلة New يجرى استدعاؤها

لتكوين أحد الكائنات ، بينما يتم استدعاء وسيلة Dispose لتحطيم هذا الكائن وإزالته من الوجود. ويجب تحطيم أى كائن عند انتهاء الحاجة إلية. والسبب فى ذلك يرجع إلى أن الكائن عند تكوينه يشغل جزءا من الذاكرة. وحيث أن الذاكرة تمثل موردا محدودا، وأن نظام التشغيل سوف يتوقف إذا لم يجد الذاكرة الكافية، لذلك يجب إزالة الكائن من الذاكرة عند انتهاء الحاجة إلية وإعادة المساحة التى كان يشغلها فى الذاكرة إلى نظام التشغيل. نعرض فيما يلى الكود الخاص بهذه الوسيلة، الذى يقوم Visual Studio.NET بتكوينه :

```
Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    If disposing Then
        If Not (components Is Nothing) Then
            components.Dispose ()
        End If
    End If
    MyBase.Dispose (disposing)
End Sub
```

يقوم هذا الكود أولا بالتخلص من أى مكونات أو أدوات تحكم توجد على النموذج ثم يستدعى وسيلة Dispose فى التصنيف الأصلى المستخدم فى تكوين الكائن، وهو التصنيف الذى تمثلة الكلمة المفتاحية MyBase.

مرشد منطقة الكود (Region Directive)

هناك جزء كبير من الكود يوجد داخل مرشد المنطقة الذى يبدأ و ينتهى كما يلى :

```
#Region " Windows Form Designer generated code "
'code
#End Region
```

يقوم Region Directive بإعلام محرر الكود (Code Editor) عن منطقة الكود التى يمكن توسيعها أو طيها. والنص الذى يلى عبارة Region، وهو " Windows Form Designer generated code"، يوضح نوعية الكود الذى تحتويه المنطقة، كما أنة يميز منطقة الكود لأن الكود قد يحتوى على أكثر من منطقة. ومع أن هذا المرشد يقوم بانتاجة Visual Studio.NET، إلا أنة يمكننا تكوين مرشد منطقة خاص بنا. وتتضح فائدة ذلك عندما يصبح الكود الذى نقوم بكتابته أكثر تعقيد وطولا. حيث أننا نستطيع إخفاء أجزاء الكود التى لا

نريد فحصها والتركيز فقط على الكود الذى نريد تحليلة.

البرمجة باستخدام الكائنات

كل شئ نقوم بعملة الآن فى Visual Basic.NET يرتبط تقريبا بالكائنات. بل إن استخدام الكائنات يقع فى المركز بالنسبة للبرمجة باستخدام Visual Basic.NET. وتشتمل البرمجة باستخدام الكائنات (Object-Oriented Programming) على الكثير من المفاهيم ويرتبط بها الكثير من التقنيات كما أنها تساهم فى تبسيط كتابة الكود وتزيد من إمكانية إعادة استخدام هذا الكود.

مفاهيم البرمجة باستخدام الكائنات

هناك عدد من المفاهيم التى تميز البرمجة باستخدام الكائنات لابد من فهمها قبل بدء استخدام هذه الطريقة فى البرمجة. تشمل هذه المفاهيم ما يلى :

- التصنيفات والكائنات.
- الحقول، الخصائص، الوسائل، والأحداث.
- التغليف، الوراثة، وتعدد أوجه الاستخدام.
- التحميل الزائد، الهيمنة، والتظليل.
- الربط المبكر والمتأخر للكائنات مع المتغيرات.
- أعضاء التصنيف المشتركة بين أمثلة التصنيف.

التصنيفات والكائنات

التصنيف (Class) هو تمثيل مجرد لشيء ما، بينما الكائن (Object) مثل قابل للاستخدام من الشيء الذى يمثله التصنيف. والاستثناء الوحيد لذلك هو أعضاء التصنيف المشتركة (shared class members)، التى يمكن استعمالها فى أمثلة التصنيف (Class Instances) وفى المتغيرات التى يجرى الإعلان عنها من نوع ذلك التصنيف.

الحقول، الخصائص، الوسائل، ولاحداث

تتكون التصنيفات من حقول (Fields)، خصائص (Properties)، وسائل (Methods)،

وأحداث (Events). تمثل الحقول والخصائص معلومات يحتوى عليها الكائن. ويمكن تمثيل الحقول بالمتغيرات لأنها تقبل القراءة والكتابة بها مباشرة. على سبيل المثال، إذا كان لدينا كائن يمثل القلم يمكننا تخزين لون القلم فى حقل باسم Color. والخصائص يتم استخراجها وضبطها مثل الحقول، غير أن ذلك يتم باستخدام إجراءات Property Get و Property Set ، التى تمكننا من إحكام الرقابة على تحديد واستخراج قيم الخصائص. تتم هذه الرقابة عن طريق استخدام الطبقة الفاصلة بين القيمة التى يتم تخزينها وبين الإجراءات التى تستخدم هذه القيمة، فى فصل البيانات والسماح لنا بمراجعة وتدقيق القيم قبل تخزينها أو استخراجها. وتمثل الوسائل الأفعال التى يمكن لكائن تنفيذها. على سبيل المثال، كائن السيارة يكون به وسيلة لبدء التشغيل، وسيلة للقيادة، ووسيلة للإيقاف. ويتم تعريف الوسائل عن طريق إضافة إجراءات من نوع Sub أو Functions إلى التصنيف. وتتسبب الإحداث فى قيام الكائنات بتنفيذ الأفعال عند وقوع الحدث. وفى نظام تشغيل الويندوز، الذى يعتبر نظام تشغيل تحركة الأحداث، يمكن أن تقع الأحداث بواسطة كائنات أخرى، تطبيقات، أو إدخالات المستخدمين بواسطة الماوس أو لوحة المفاتيح.

التغليف، الوراثة، تعدد أوجه الاستخدام

تمثل الحقول، الخصائص، الوسائل، والأحداث نصف الصورة بالنسبة للبرمجة باستخدام الكائنات. النصف الأخر يشتمل على ثلاثة مفاهيم مهمة: التغليف (Encapsulation)، الوراثة (Inheritance)، و تعدد أوجه الاستخدام (Polymorphism).

يعنى التغليف أن مجموعة من الخصائص ذات العلاقات، الوسائل، والأعضاء الأخرى يتم معاملتها على أنها وحدة واحدة أو كائن. ويمكن أن تتحكم الكائنات فى كيفية تغيير الخصائص وتنفيذ الوسائل. على سبيل المثال، يمكن أن يقوم الكائن بتدقيق القيم قبل تغيير الخصائص. كما تساعدنا عملية التغليف على تغيير التنفيذ فى تاريخ لاحق عن طريق توفير إمكانية إخفاء تفصيلات التنفيذ للكائنات، وهو ما يسمى إخفاء البيانات.

وتعنى الوراثة ببساطة إمكانية تكوين تصنيفات جديدة بالاعتماد على تصنيفات قائمة. حيث يقوم التصنيف الجديد بوراثة خصائص، وسائل، وأحداث من التصنيف الأصلى، ويمكن تعديل التصنيف الجديد عن طريق إضافة خصائص ووسائل أخرى. على سبيل

المثال، يمكن تكوين تصنيف جديد باسم شاحنة بالاعتماد على تصنيف للسيارة. يقوم التصنيف الجديد بوراثة خاصية اللون من تصنيف السيارة ويمكنة أن يحتوى على خصائص إضافية.

تعدد أوجه الاستخدام يعنى وجود تصنيفات متعددة يمكن استخدامها مكان بعضها البعض، مع أن كلا منها ينتج نفس الخصائص والوسائل بطرق مختلفة. وتعتبر هذه العملية ضرورية بالنسبة للبرمجة باستخدام الكائنات لأنها تسمح باستخدام البنود التى لها نفس الاسم، بغض النظر عن نوع الكائن الجاري استخدامه فى تلك اللحظة. على سبيل المثال، باستخدام تصنيف السيارة، تسمح عملية Polymorphism للمبرمج بتعريف وسائل مختلفة لبدء تشغيل المحرك لأي عدد من التصنيفات المشتقة. وسيلة بدء تشغيل السيارة التى تعمل بالديزل يمكن أن تكون مختلفة بالكامل عن وسيلة بدء التشغيل التى لها نفس الاسم فى التصنيف الأصلى. ويمكن للإجراءات والوسائل الأخرى استخدام وسيلة التشغيل فى التصنيفات المشتقة من تصنيف السيارة الأصلى بنفس الطريقة، بغض النظر عن نوع كائن السيارة المستخدم فى ذلك الوقت.

التحميل الزائد، الهيمنة، و التظليل

استخدام التحميل الزائد (Overloading)، الهيمنة (Overriding)، والتظليل (Shadowing) تعتبر كلها مفاهيم متشابهة ويمكن أن تثير اضطراب الفهم بسهولة. ومع أن كلا من التقنيات الثلاثة تمكن من تكوين أعضاء تصنيفات لها نفس الاسم، إلا أن هناك بعض الاختلافات المهمة بينها.

- تستخدم تقنية Overloading فى تكوين أعضاء تصنيف تقوم بتقديم إصدارات مختلفة من خاصية أو وسيلة ويكون لها نفس الاسم، ولكنها تقبل عدد مختلف من المعاملات، أو تقبل معاملات من أنواع مختلفة.
- الخصائص والوسائل المهيمنة تستخدم لتحل محل خاصية أو وسيلة موروثة غير مناسبة فى التصنيف المشتق. ويجب أن يقبل أعضاء التصنيف المهيمنة نفس نوع البيانات وعدد المعاملات. ويتم وراثة الأعضاء المهيمنة بواسطة التصنيفات المشتقة.
- الأعضاء المظللة تستخدم فى تكوين أعضاء محلية بدلا من أعضاء ذات نطاق رؤية

أوسع (Broader Scope). ويمكن أن يقوم أى نوع بوضع أي نوع أخر فى الظل. على سبيل المثال، يمكننا الإعلان عن خاصية لتظليل وسيلة موروثة لها نفس الاسم. ولا يمكن وراثة الأعضاء المظللة.

ربط الكائنات المبكر والمتأخر

يقصد بربط الكائنات عملية تخصيص كائن بعد تكوينه، لأحد المتغيرات من نوع الكائن أو من نوع Object. وهناك نوعان من الربط بين الكائن وبين المتغير الذى يمثل الكائن يقوم بها برنامج ترجمة الكود (Compiler) فى Visual Basic. النوع الأول، وهو الربط المبكر (Early Binding) الذى يتم عند تخصيص الكائن لمتغير سبق تحديده من نوع معين عند الإعلان عنة. يؤدى هذا الربط المبكر إلى قيام برنامج الترجمة بتخصيص ذاكرة للكائن قبل تنفيذ التطبيق. يتضح هذا النوع من الربط فى الكود التالى :

```
Imports System.IO

Dim FS As FileStream
FS = New FileStream("C:\tmp.txt", FileMode.Open)
```

فى المثال السابق، تم الإعلان عن متغير من نوع FileStream ثم ربطة مع المتغير F5. يعتبر التخصيص الذى تم للمتغير FS من نوع الربط المبكر للكائنات نظرا لأن FileStream يعتبر نوع محدد من الكائنات.

وعلى العكس من ذلك، يعتبر الربط متأخرا (Late Binding) بالنسبة للكائن عند تخصيصه لمتغير سبق الإعلان عنة من نوع كائن (Object). الكائنات من هذا النوع يمكن أن تحتفظ بمراجع أي كائن أخر بغض النظر عن نوعة، ولكن تنقصها الكثير من المزايا الخاصة بالربط المبكر للكائنات. على سبيل المثال، الكود التالى يعلن عن متغير من نوع Object لإستخدامه فى الاحتفاظ بكائن ناتج من تنفيذ دالة CreateObject :

```
Option Strict Off ' Option Strict Off allows late binding.
Public Class Form1
   Inherits System.Windows.Forms.Form
#Region " Windows Form Designer generated code "

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
```

```
System.EventArgs) Handles Button1.Click
        TestLateBinding ()
    End Sub
Sub TestLateBinding ()
     Dim xlApp As Object
     Dim xlBook As Object
     Dim xlSheet As Object
     xlApp = CreateObject("Excel.Application")

     xlBook = xlApp.Workbooks.Add
     xlSheet = xlBook.Worksheets (1)
     xlSheet.Activate()
     xlSheet.Application.Visible = True ' Show the
     application.
     xlSheet.Cells (2, 2) = "This is column B row 2"
End Sub

End Class
```

فى المثال السابق، تم استخدام إجراء TestLateBinding لتكوين كائن تطبيق Excel وتشغيله. وفيما يلى تحليل للكود الذى يحتوى علية هذا الإجراء :

- عبارة xlApp = CreateObject ("Excel.Application"). تستخدم دالة CreateObject لتكوين كائن تطبيق Excel وتخصيصه لمتغير xlApp باستخدام الربط المتأخر.
- عبارة xlBook = xlApp.Workbooks.Add. تقوم بإضافة كتاب عمل إلى مجموعة كتب تطبيق Excel السابق تكوينه وتخصيصها للمتغير xlBook باستخدام الربط المتأخر.
- عبارة xlSheet = xlBook.Worksheets (1). تقوم بإضافة ورقة عمل إلى مجموعة أوراق العمل بكتاب تطبيق Excel السابق تكوينه ثم تخصيصها للمتغير xlSheet باستخدام الربط المتأخر.
- عبارة xlSheet.Activate ()، تقوم بتنشيط صفحة Excel السابق إضافتها.
- عبارة xlSheet.Application.Visible = True، تقوم بإظهار التطبيق.
- عبارة "xlSheet.Cells (2, 2) = "This is column B row 2. تقوم بوضع بعض النص فى الصف الثانى من الصفحة.

ويجب استخدام كائنات الربط المبكر كلما كان ذلك ممكنا، لأنها تمكن المترجم (Compiler) من إنتاج تطبيقات أكثر فاعلية. ومن مزايا كائنات الربط المبكر أنها أسرع بطريقة جوهرية من كائنات الربط المتأخر وتجعل الكود سهل فى القراءة والصيانة عن طريق تحديد أنواع الكائنات المستخدمة. ومن مزايا الربط المبكر أيضا، أنها تمكننا من الاستفادة من سمات مفيدة، مثل إكمال الكود أتوماتيكيا والمساعدة الديناميكية، لأن بيئة التطوير المتكاملة (IDE) فى Visual Studio.NET يمكنها تقرير نوع الكائن الذى نعمل معه بالضبط أثناء تحرير الكود. ويؤدى الربط المبكر إلى تخفيض عدد ودرجة أخطاء وقت التشغيل لأنها تمكن المترجم من عرض الأخطاء أثناء ترجمة البرنامج. ويجب ملاحظة أن الربط المتأخر يمكن إستخدامة فقط للوصول إلى أعضاء التصنيف التى يتم إعلانها عامة (Public).

أعضاء التصنيف المشتركة

الأعضاء المشتركة (Shared Members) هى خصائص، إجراءات، و حقول تشارك فيها جميع أمثلة التصنيف. وتظهر فائدة الخصائص والحقول عند وجود معلومات تعتبر جزءا من التصنيف، ولا يختص بها أحد أمثلة التصنيف بمفردة. فى الوضع الطبيعي توجد الحقول والخصائص مستقلة فى كل مثل من أمثلة التصنيف، مما يترتب علية أن تغيير قيمة حقل أو خاصية فى أحد أمثلة التصنيف لا يؤثر على قيم نفس الحقول والخصائص فى الأمثلة الأخرى للتصنيف. من ناحية أخرى، عند تغيير قيمة حقل أو خاصية مشتركة فى أحد أمثلة التصنيف، فإن قيمة هذا الحقل أو الخاصية تتغير فى جميع الأمثلة الأخرى للتصنيف. بهذه الطريقة، تعمل الحقول والخصائص المشتركة مثل المتغيرات العامة التى يمكن الوصول إليها فقط من داخل أمثلة التصنيف.

الإجراءات المشتركة هى وسائل بالتصنيف غير مرتبطة بمثل محدد من هذا التصنيف. ويمكن استدعاء الإجراء المشترك على أنة وسيلة فى أحد كائنات التصنيف، أو مباشرة من التصنيف ذاته. على هذا الأساس، يمثل الإجراء المشترك استثناء للقاعدة التى تفرض تكوين مثل من التصنيف لكى نستطيع إستخدامة.

لتوضيح استخدام الأعضاء المشتركة، يقوم التمرين التالى بتكوين حقل بأحد الأمثلة، حقل مشترك، ووسيلة مشتركة. ولتنفيذ هذا التمرين، نتبع الخطوات التالية :

١. فى القائمة الرئيسية، نختار File، نشير إلى New ثم ننقر Project.

٢. فى الجانب الأيسر من مربع حوار New Project نختار Visual Basic Projects، وفى الجانب الأيمن نختار Console Application. نطبع اسما للمشروع فى مربع Name أو نترك الاسم الإفتراضى بدون تغيير، كما نحدد المسار الذى سوف يتم حفظ المشروع به فى مربع Location ثم ننقر OK.

٣. فى مربع تحرير الكود ندخل الكود التالى :

```
Module Module1
   Public Class ShareClass
      Public InstanceValue As String
      Public Shared SharedValue As String
      Public Shared Sub ShareMethod ()
         MsgBox ("This is a shared method.")
      End Sub
   End Class
   Sub TestShared ()
      Dim Shared1 As New ShareClass ()
      Dim Shared2 As New ShareClass ()
      Shared1.SharedValue = "Share Value 1"
      Shared2.SharedValue = "Share Value 2"
      MsgBox ("The value of the shared field in the
            first instance" & _
        "is: " & Shared1.SharedValue)
      MsgBox ("The value of the shared field in the
second instance" & "is: " & _ Shared2.SharedValue)

      ShareClass.ShareMethod()
   End Sub

   Sub Main()
      TestShared()
   End Sub

End Module
```

٤. نضغط على مفتاح F5 لتشغيل التطبيق. يترتب على ذلك ، عرض ثلاثة رسائل. الرسالة الأولى توضح أن قيمة الحقل المشترك فى المثل الأول هى القيمة التى تم تغييرها لنفس الحقل فى المثل الثانى. والرسالة الثانية تبين القيمة التى تم إدخالها فى الحقل المشترك بالمثل الثانى. والرسالة الثالثة تفيد بأن وسيلة

ShareMethod التى تم استدعائها باستخدام اسم التصنيف وليس أسم أحد أمثلته، هو وسيلة مشتركة.

يشتمل إجراء TestShared على تكوين مثلين من تصنيف ShareClas ، ثم تعديل قيمة حقل SharedValue فى كلا المثلين. وعند تغيير قيمة الحقل المشترك SharedValue فى المثل الثانى من التصنيف، أدى ذلك الى تغيير قيمة نفس الحقل فى المثل الأول.

ومع أننا فى هذا المثال قد استخدمنا متغيرات الكائنات للوصول إلى الأعضاء المشتركة فى التصنيف، إلا أن الإجراء الصحيح فى هذه الحالة هو استخدام اسم التصنيف مباشرة للوصول إلى تلك الأعضاء المشتركة. على سبيل المثال، نستخدم ShareClass.SharedValue "Share Valu2" = بدلا من "Shared2.SharedValue = "Share Value2 فى الكود السابق.

دور التصنيفات فى البرمجة باستخدام الكائنات

التصنيفات (Classes) هى الأساس الذى تقوم علية البرمجة باستخدام الكائنات لأنها تمثل القوالب التى تستنسخ منها الكائنات. ويقوم التصنيف بوضع المعلومات ذات العلاقة فى مجموعة ومعاملتها على أنها وحدة واحدة والتحكم فى رؤيتها والوصول إليها بواسطة الإجراءات الأخرى. ويمكن للتصنيفات وراثة التصنيفات الأخرى وإعادة استخدام الكود الذى تحتوى علية. وتعامل التصنيفات فى Visual Basic .NET على أنها أنواع للبيانات ويتم استخدامها فى عبارات الإعلان عن الكائنات.

وتماثل التصنيفات وحدات الكود (Modules) فى أنها أنوع تخفى داخلها البنود التى تتكون منها، إلا أنها تختلف عنها فى كيفية الوصول إلى هذه البنود من قبل الإجراءات الأخرى. والفرق الأساسى بين التصنيف وبين وحدة الكود هو أن التصنيف يمكن استنساخه ولا يمكن القيام بذلك مع وحدات الكود. ونظرا أن وحدة الكود يوجد منها نسخة واحدة، لذا يترتب على تغيير متغير عام فى وحدة كود حصول أى جزء من البرنامج على القيمة الجديدة عند قراءة هذا المتغير بعد تغييره. من ناحية أخرى ، نجد أن كل كائن من كائنات التصنيف يحتوى على نسخة منفصلة من بيانات التصنيف. ويختلف التصنيف عن وحدة الكود أيضا فى أنة يمكنة استخدام واجهات استخدام (Interfaces). كما يختلف نطاق رؤية الأعضاء بين التصنيف وبين وحدة الكود. الأعضاء التى يتم تعريفها داخل تصنيف، يقع نطاق رؤيتها داخل أحد كائنات التصنيف، كما أنها تتواجد فقط خلال فترة

حياة الكائن. يترتب على ذلك، أننا لا نستطيع الوصول إلى أحد أعضاء التصنيف من خارجة إلا بعد تأهيله باسم أحد كائنات التصنيف. على الناحية الأخرى، تعتبر الأعضاء التى يتم الإعلان عنها داخل وحدات الكود القياسية ، أعضاء مشتركة بطبيعتها. يعنى ذلك أن المتغيرات العامة (Public Variables) فى وحدة كود، يمكن رؤيتها من أى مكان فى المشروع وتتواجد طول فترة بقاء التطبيق.

تعريف التصنيفات

نستخدم Class Module فى قائمة Project بالقائمة الرئيسية فى Visual Studio.NET لتعريف تصنيف جديد. ويمكن أيضا تعريف تصنيف جديد باستخدام محرر الكود فى Visual Basic .NET عن طريق طباعة كلمة Class متبوعة باسم التصنيف الجديد. وليس هناك حاجة إلى إضافة عبارة نهاية التصنيف End Class، لأن محرر الكود يقوم تلقائيا بإضافة هذه العبارة. يوضح التمرين التالى كيفية استخدام Class Module لتعريف التصنيفات، التى يمكن استخدامها بعد ذلك فى تكوين الكائنات. كما يبين كيفية تكوين الخصائص والوسائل وكيفية بدء تشغيل الكائنات.

إدخال تعريف التصنيف

١. نفتح مشروع تطبيق ويندوز بالنقر على New بقائمة File ثم النقر على Project. يترتب على ذلك عرض مربع حوار New Project.

٢. نختار Windows Application من قائمة قوالب مشروعات Visual Basic ، مما يؤدى إلى فتح المشروع الجديد.

٣. نضيف تصنيف جديد إلى المشروع بالنقر على Add Class فى قائمة Project. يترتب على ذلك ظهور مربع حوار Add New Item.

٤. نجعل اسم الوحدة الجديدة StateInfo.vb ثم ننقر Open. يؤدى ذلك إلى ظهور نافذة تحرير الكود الخاص بالتصنيف الجديد.

٥. لتبسيط الوصول إلى التصنيفات المسجلة بالنظام، نضيف عبارة Imports فى قمة الكود الذى يحتوى على عبارة Class:

```
Imports Microsoft.Win32
```

٦. نعرف ثلاثة حقول خاصة (Private Fields) فى التصنيف بوضع الكود التالى بين

عبارة Class وعبارة End Class :

```
Private pVal As String
Private KeyName As String
Private SubKeyName As String
```

لقد تم الإعلان عن هذه الحقول الثلاثة باستخدام كلمة Private، مما يعنى إمكانية استخدامها فقط داخل التصنيف. ويمكن جعل الوصول إلى الحقول ممكنا من خارج التصنيف عن طريق تراخيص الوصول (Access Modifiers) ، مثل كلمة Public.

٧. نعرف خاصية بالتصنيف عن طريق إضافة الكود التالى :

```
Public Property LastFile() As String
  Get
    Return pVal
  End Get
  Set(ByVal Value As String)
    pVal = Value
  End Set
End Property
```

٨. نعرف وسائل خاصة بالتصنيف عن طريق إضافة الكود التالى :

```
Sub SaveStateInfo()
   Dim aKey As RegistryKey
   aKey = Registry.CurrentUser.CreateSubKey(KeyName)
  aKey.SetValue(SubKeyName, pVal)
End Sub


Sub GetStateInfo()
   Dim aKey As Object
   Dim myRegKey As RegistryKey = Registry.CurrentUser
  Try
   myRegKey = myRegKey.OpenSubKey(KeyName)
   Dim oValue As Object = myRegKey.GetValue(SubKeyName)
   pVal = CStr(oValue)
  Catch
   pVal = ""
  End Try
 End Sub
```

٩. نعرف إجراء بناء يحتوى بة معاملات خاص ببناء كائنات التصنيف باضافة

إجراء يسمى Sub New:

```
Sub New(ByVal RegistryKeyName As String, _
      ByVal RegistrySubKeyName As String)
  KeyName = RegistryKeyName
  SubKeyName = RegistrySubKeyName
  MyClass.GetStateInfo()
End Sub
```

١٠. نعرف إجراء الهدم الخاص بالتصنيف عن طريق إضافة الكود التالى :

```
Protected Overrides Sub Finalize()
  SaveStateInfo()
  MyBase.Finalize()
End Sub
```

يقوم إجراء الهدم Finalize بحفظ قيمة الخاصية فى مسجل النظام بعد توارى التصنيف عن بؤرة التركيز.

إضافة زر لاختبار التصنيف

١. نتحول إلى مشهد تصميم النموذج بالنقر بزر الماوس الأيمن على اسم النموذج الإفتراضى فى مربع Solution Explorer ، ثم النقر على View Designer. يترتب على ذلك ظهور النموذج الأساسى.

٢. نضيف متحكم Button إلى النموذج وننقر نقرا مزدوجا علية لعرض الكود الخاص بإجراء معالجة حدث Button1_Click. نضيف الكود التالى لاستدعاء إجراء الاختبار :

```
Dim SI As New StateInfo("Software\StateInfo", "PropertyValue")
If Len(SI.LastFile) > 1 Then
  MsgBox("The value of the property LastFile is: " _
      & SI.LastFile)
Else
  MsgBox("The LastFile property has not been set.")
End If
SI.LastFile = "C:\BinaryFile.txt"
SI.SaveStateInfo()
```

تشغيل التطبيق

١. نضغط مفتاح F5 لتشغيل التطبيق.

٢. ننقر الزر الذى على النموذج لاستدعاء إجراء الاختبار. عند القيام بذلك للمرة

الأولى، سوف تظهر رسالة تفيد بأن خاصية LastFile لم يتم ضبطها.

٣. ننقر على OK للتخلص من مربع الرسالة. يقوم إجراء Button1_Click بضبط قيمة خاصية LastFile ويستدعى وسيلة SaveState. وحتى فى حالة عدم استدعاء وسيلة SaveState صراحة من داخل إجراء Button1_Click، فإنها سوف تستدعى فى إجراء Finalize تلقائيا بعد إقفال نموذج بدء التطبيق.

٤. ننقر على زر النموذج مرة أخرى، مما يترتب علية عرض رسالة "The value of the property LastFile is C:\BinaryFile.txt".

٥. نقفل النموذج الأساسى فى التطبيق ثم نعيد تشغيل البرنامج مرة أخرى بالضغط على مفتاح F5. يتكون بذلك كائن مبنى على التصنيف، ويقو إجراء Sub New به باستدعاء إجراء GetStateInfo الذى يستعيد قيمة الخاصية من مسجل النظام (Registry). وتعرض رسالة "The value of the property LastFile is C:\BinaryFile.text" مرة أخرى عند النقر على الزر.

تكوين وهدم الكائنات

تبدأ حياة الكائنات عند تكوين مثل من أحد التصنيفات باستخدام الكلمة المفتاحية New. وتتطلب الكائنات الجديدة فى الغالب تنفيذ بعض المهام المتعلقة بالإعداد قبل استخدامها للمرة الأولى. وتشمل مهام الإعداد الشائعة فتح الملفات، الاتصال مع قاعدة بيانات، وقرءاه قيم مفاتيح مسجل النظام (Registry Keys). ويتحكم Visual Basic.NET فى إعداد الكائنات الجديدة باستخدام إجراءات تسمى Constructors.

وتنتهي حياة الكائن عندما يخرج عن نطاق الرؤية ويتم التخلص منة بواسطة مدير تشغيل الكود (Common Language Runtime). ويتحكم Visual Basic.NET فى عملية التخلص من الكائنات وتحرير موارد النظام التى تستخدمها باستخدام إجراءات تسمى Destructors.

إجراء Sub New وإجراء Sub Finalize

تقوم هذه الإجراءات فى Visual Basic.NET بإعداد وهدم الكائنات. ويعمل إجراء Sub New مرة واحدة فقط عند بدء تكوين الكائن ، كما لايمكن استدعاء هذا الإجراء صراحة إلا فى السطر الأول من كود إجراء بناء أخر بنفس التصنيف أو فى تصنيف مشتق من التصنيف

صاحب الإجراء. إضافة إلى ذلك ، يتم تشغيل الكود الخاص بهذا الإجراء قبل أى كود أخر فى التصنيف. ويقوم Visual Basic .NET ضمنيا بتكوين إجراء بناء وقت التشغيل عند عدم تحديده فى التصنيف. وقبل التخلص من الكائنات، يقوم مدير تشغيل الكود (CLR) باستدعاء وسيلة Finalize التى تحتوى على إجراء Sub Finalize بالكائن. ويمكن أن تحتوى وسيلة Finalize على كود يجب تنفيذه قبل هدم الكائن مباشرة، مثل إقفال الملفات، وحفظ معلومات الحالة. ويمكن إهمال تكوين وسيلة Sub Finalize عند عدم الحاجة إليها. ولا يمكن استدعاء وسيلة Finalize إلا من التصنيف الذى تتبعة أو من تصنيف موروث منة. ويقوم النظام باستدعاء وسيلة Finalize تلقائيا عند هدم أحد الكائنات، ولهذا لايجب استدعاء هذه الوسيلة صراحة خارج وسيلة Finalize بتصنيف مشتق من التصنيف الذى تتبعة.

استخدام إجراءات البناء والهدم

تقوم إجراءات البناء والهدم (Constructors and Destructors) بالتحكم فى بناء وهدم الكائنات. ولتكوين إجراء بناء لأحد التصنيفات، نقوم بتكوين إجراء باسم Sub New فى أى مكان بالتصنيف. ولتكوين إجراء بناء يحتوى على معاملات، نحدد أسماء ونوع بيانات المعاملات بإجراء Sub New مثلما نفعل مع أى إجراء أخر، كما يتضح من الكود التالى :

```
Sub New (ByVal sString As String, iInt As Integer)
```

وعندما يتم تعريف تصنيف مشتق من تصنيف أخر، يجب أن يحتوى السطر الأول فى إجراء بناء التصنيف المشتق على استدعاء إجراء البناء فى التصنيف الأساسى، إلا إذا كان التصنيف الأساسى به إجراء بناء يمكن الوصول إلية ولا يحتوى على معاملات. ويكون استدعاء إجراء البناء فى تصنيف أساسى كما يلى :

```
MyBase.New (sString)
```

وعندما لا يحتوى إجراء البناء الأساسى على معاملات، يصبح استدعاؤه فى التصنيف المشتق اختياريا. وفى حالة عدم استدعائه، يقوم Visual Basic .NET بعملية الاستدعاء تلقائيا. وبعد كتابة الكود اللازم لاستدعاء إجراء بناء التصنيف الأصلى، يمكننا إضافة أى كود إعداد فى إلى إجراء البناء فى التصنيف المشتق. ويمكن أن يقبل إجراء البناء معاملات عندما يتم استدعاؤه بالصورة التى يحتوى فيها على المعاملات(Parameterized Constructors). ويقوم بتمرير هذه المعاملات الإجراء الذى يستدعى إجراء البناء، كما يتضح مما يلى :

```
Dim tstObject As New ThisClass (X)
```

خصائص، حقول، ووسائل التصنيفات

تمثل الحقول والخصائص معلومات عن الكائن الذى توجد به، بينما تمثل الوسائل الأفعال التى يمكن أن يقوم بها هذا الكائن. فيما يلى نوضح كيفية التعامل مع هذه البنود.

إضافة الخصائص والحقول إلى التصنيف

يمكن استخدام كلا من الحقول والخصائص لتخزين معلومات أحد الكائنات. ويتم الإعلان عن الحقول والخصائص بطريقة مختلفة داخل التصنيف. فبينما تعتبر الحقول متغيرات عامة (Public Variables) يكشف عنها التصنيف ، تستخدم الخصائص إجراءات Property للتحكم فى كيفية تخزين البيانات وكيفية استعادتها بعد التخزين.

لإضافة حقل الى تصنيف ، نعلن عن متغير عام فى تعريف التصنيف ، كما فى الكود التالى :

```
Class ThisClass
  Public ThisField As String
End Class
```

ولإضافة خاصية إلى تصنيف ، نتبع الخطوات التالية :

١. نعلن عن متغير محلى (Local Variable) داخل التصنيف لتخزين قيمة الخاصية. وتعتبر هذه الخطوة مهمة لأن الخاصية لا تقوم بحجز أى مساحة فى الذاكرة. وللمحافظة على قيم الخصائص من التعديل المباشر ، نستخدم الترخيص Private عند تعريف متغير الخاصية.

٢. نستهل الإعلان عن الخصائص باستخدام التراخيص ، مثل Public أو Shared. نستخدم كلمة Property للإعلان عن اسم الخاصية ، ثم الإعلان عن نوع البيانات التى تخزنها الخاصية و العائدة منها.

٣. نقوم بتعريف إجراءات Get و Set داخل تعريف الخاصية. يستخدم إجراء Get لإعادة قيمة الخاصية، وهو يشبة تركيبات الدوال (Functions). ولا تقبل هذه الإجراءات معاملات ويمكن استخدامها لإعادة قيم المتغيرات المحلية الخاصة التى يتم الإعلان عنها داخل التصنيف. وتستخدم إجراءات Set فى ضبط قيمة الخاصية. ويحتوى هذا الإجراء على معامل يطلق علية فى العادة Value، ويكون من نفس نوع بيانات الخاصية ذاتها. ولتغيير قيمة الخاصية ، يتم تمرير Value إلى

إجراء Set حيث يمكن تدقيقها وتخزينها فى متغير محلي.

٤. ولإقفال إجراءات Get و Set، نستخدم عبارات End Get و End Set المناسبة.

٥. لإقفال وحدة كود الخاصية، نستخدم عبارة End Property. ويمكن توجيه Visual Studio.NET لتكوين هيكل وحدة الكود الأساسية التى تمثل خاصية عن طريق إدخال كلمة Property متبوعة باسم الخاصية ثم كلمة As متبوعة بنوع بيانات الخاصية داخل تعريف التصنيف فى نافذة تحرير الكود. على سبيل المثال، إذا كان اسم الخاصية MyPropery ونوع البيانات Integer، نحصل على الكود التالى:

```
Property MyPropery() As Integer
       Get

       End Get
       Set(ByVal Value As Integer)

       End Set
 End Property
```

ولتوضيح كيفية تكوين الخصائص، نعرض الكود التالى الذى يستخدم المتغير الخاص m_Variable لتخزين قيمة الخاصية :

```
Class MyClass
  Private m_Variable As String
  Public Property MyProperty () As String
    Get
      Return m_Variable
    End Get
    Set(ByVal Value As String)
      m_Variable = Value
    End Set
  End Property
End Class
```

عند تكوين مثل من هذا التصنيف وضبط قيمة خاصية MyProperty السابق إيضاحها، يتم استدعاء إجراء Set وتمرير القيمة من خلال معامل Value، ثم تخزين القيمة فى المتغير m_Variable. وعند استعادة قيمة هذه الخاصية، يجرى استدعاء إجراء Get الذى يعيد إلينا القيمة المخزنة فى المتغير m_Variable.

المفاضلة بين الحقول والخصائص

مع أن كلا من الحقول والخصائص تستخدم لنفس الوظيفة وهى حفظ البيانات الخاصة بالتصنيف و استعادتها، إلا أن هناك ظروفا تستدعى استخدام الخصائص وأخرى يكون من المفضل فيها استخدام الحقول.

نستخدم الخصائص عند توفر الشروط التالية :

- عند الحاجة إلى التحكم فى وقت وكيفية ضبط واسترجاع القيمة.
- عندما يتبع الخاصية مجموعة من القيم التى تتطلب المراجعة والتدقيق.
- عندما يترتب على ضبط القيمة تغيير محسوس فى حالة الكائن، مثل إخفاء وإظهار الكائن.
- عندما يترتب على ضبط الخاصية تغيير فى المتغيرات الداخلية الأخرى أو تغيير قيم الخصائص الأخرى.
- عندما يتطلب تغيير أو استعادة القيمة تنفيذ عدد من الخطوات.

ونستخدم الحقول عند توفر الشروط التالية :

- عندما تكون القيمة من النوع الذى تتم مراجعته ذاتيا. على سبيل المثال، عند تخصيص قيمة لمتغير من نوع القيم المنطقية (Boolean Variable)، تكون القيمة إما True أو False وإدخال أى قيمة أخرى يترتب علية إطلاق خطأ أو التحويل التلقائي للقيمة الصحيحة.
- عندما تكون قيمة الخاصية صحيحة إذا وقعت فى نطاق القيم التى يدعمها نوع البيانات المستخدم.
- عندما يكون نوع بيانات الخاصية هو السلسلة (String) ولا يوجد قيود على حجم أو قيمة السلسلة.

استخدام الوسائل فى التصنيفات

يمكن تعريف الوسائل بأنها الإجراءات الفرعية (Sub) والدوال (Functions) العامة التى يتم الإعلان عنها داخل التصنيف. يوضح الكود التالى كيفية إضافة دالة عامة إلى تصنيف تتطلب معامل من نوع Decimal وتعيد قيمة من نوع Double:

```
Public Function MyFunction (ByVal Amount As Decimal) As Double

End Function
```

الوسائل المشتركة (Shared Methods)

يمكن استدعاء الوسائل المشتركة مباشرة باستخدام متغير تصنيف بدون الحاجة إلى تكوين مثل من التصنيف أولا. وتظهر فائدة الوسائل المشتركة عندما لا نريد ربط الوسيلة بكائن معين من كائنات التصنيف. يوضح المثال التالى أستخدم الوسائل المشتركة فى تصنيف وكيفية استدعائها :

```
Class ShareClass
     Shared Sub SharedSub ()
       MessageBox.Show ("Shared method.")
     End Sub
End Class

Sub Test()
   Dim S As ShareClass
   S.SharedSub
End Sub
```

حماية تفصيلات التصنيف

أعضاء التصنيف المستخدمة للخدمة داخل هذا التصنيف يجب حمايتها باستخدام كلمات Private، Protected، أو Friend. استخدام هذه الكلمات يؤدى إلى تقييد الوصول إلى تلك الأعضاء ومنع استخدامها بواسطة المبرمجين الآخرين. كما تسمح لنا بعمل التغييرات المستقبلية بها بدون التأثير على الكود الذى يستخدم هذه الكائنات.

ويعتبر إخفاء تفصيلات تنفيذ أحد الكائنات، وجه أخر من أوجه مفهوم التغليف (Encapsulation). و يسمح لنا مفهوم التغليف بتحسين أداء الإجراءات، أو التغيير الكامل لكيفية تطبيق هذه الإجراءات، بدون تغيير الكود الذى يستخدمها.

العلاقة بين الوسائل والخصائص

تتشابه الوسائل والخصائص فى أن كليهما يتم تنفيذه على أنة إجراء يقبل معاملات. وبصفة عامة تقوم الخصائص بوظيفة محددة هى حفظ واسترجاع البيانات الخاصة بأحد الكائنات، بينما تقوم الوسائل بتنفيذ الأعمال التى يطلب من الكائن تنفيذها. ومع أن

التركيب اللغوى الخاص باستعادة معلومات من خاصية بها معاملات يماثل تقريبا التركيب المستخدم مع وسيلة منفذة فى صورة دالة، إلا أن التركيب اللغوى الخاص بتغيير قيمة يكون مختلفا قليلا. على سبيل المثال، عند تنفيذ أحد أعضاء تصنيف على أنة خاصية ، نستخدم كود مماثل للكود التالى :

```
ThisObject.ThisProperty (Index) = NewValue
```

وعند تنفيذ نفس عضو التصنيف على أنة وسيلة ، نستخدم كود مماثل لما يلى :

```
ThisObject.ThisProperty (Index, NewValue)
```

الخصائص الافتراضية (Default Properties)

الخاصية التى تقبل معاملات يمكن الإعلان عنها على أنها الخاصية الافتراضية للتصنيف. ويمكن تعريف الخاصية الافتراضية بأنها الخاصية التى يستعملها Visual Basic.NET عندما لا يتم تسمية خاصية محددة للكائن. وتظهر فائدة الخصائص الافتراضية فى أنها تختصر الكود المستخدم بسبب حذف أسماء الخصائص التى يتكرر استخدامها. وأفضل الخيارات لتحديد الخصائص الافتراضية هو استخدام الخصائص التى تقبل المعاملات ويتكرر استخدامها. وتنطبق القواعد التالية على الخصائص الافتراضية :

- أى نوع يمكن أن يكون به خاصية افتراضية واحدة ، بما فى ذلك الخصائص الموروثة من التصنيف الأصلى.
- لايمكن أن تكون الخاصية الافتراضية مشتركة (Shared) أو خاصة (Private).
- إذا كانت الخاصية الافتراضية موجودة فى أكثر من صورة بنفس الاسم (Overloaded) ، فإن كل صور هذه الخاصية يجب أن تستخدم كلمة Default فى مقدمتها.
- الخاصية الافتراضية يجب أن تقبل معامل واحد على الأقل.
- لتوضيح استخدام الخصائص الافتراضية، يعلن المثال التالى عن خاصية افتراضية تحتوى على مصفوفة بيانات من نوع السلاسل (Strings):

```
Class Class2
  Private PropertyValues As String ()
  Default Public Property Prop1 (ByVal Index As Integer) As String
    Get
      Return PropertyValues (Index)
```

```
      End Get
      Set (ByVal Value As String)
        If PropertyValues Is Nothing Then
          ReDim PropertyValues (0)
        Else
          ReDim Preserve PropertyValues (UBound (PropertyValues) + 1)
        End If
          PropertyValues (Index) = Value
      End Set
    End Property
  End Class
```

وهناك طريقتان لاستخدام الخاصية الافتراضية. الطريقة التقليدية يستخدم فيها اسم الخاصية مؤهلا باسم الكائن، كما يوضحها الكود التالى :

```
Dim C As New Class2 ()
C.Prop1 (0) = "Value One"
MessageBox.Show (C.Prop1 (0))
```

الطريقة الثانية يستخدم فيها اسم التصنيف مباشرة، كما يتضح من الكود التالى :

```
C (1) = "Value Two"
MessageBox.Show(C (1))
```

التحميل الزائد لأعضاء التصنيف (Overloading)

يعنى التحميل تكوين أكثر من إجراء عادى، إجراء بناء مثل من تصنيف، أو خاصية فى تصنيف له نفس الاسم مع اختلاف عدد وأنواع المعاملات المطلوبة. وتظهر فائدة التحميل بصفة خاصة عند الحاجة إلى إجراءات لها نفس الاسم ولكنها تعمل على أنواع بيانات مختلفة. على سبيل المثال، التصنيف الذى يمكن أن يعرض أنواع بيانات مختلفة يمكن أن يحتوى على إجراءات عرض تماثل الإجراء التالى :

```
Overloads Sub Display (ByVal theChar As char)

End Sub

Overloads Sub Display (ByVal theInteger As Integer)

End Sub

Overloads Sub Display(ByVal the Double As Double)
```

```
End Sub
```

بدون استخدام الكلمة المفتاحية Overloads، نحتاج إلى تكوين إجراءات بأسماء مختلفة. وتجعل عملية التحميل استخدام الخصائص والوسائل أسهل لأنها توفر لنا اختيار نوع البيانات الذى نريد استخدامه. على سبيل المثال، يمكننا استدعاء نفس الإجراء باستخدام معاملات ذات أنواع بيانات مختلفة. وفى وقت التشغيل يقوم Visual Basic.NET باستدعاء الإجراء المناسب لنوع بيانات المعاملات المستخدمة.

التمرين التالى يقوم بتكوين وسائل محملة تقبل معاملات من نوع String أو Decimal تمثل المبيعات بالدولارات وتعيد مبلغ الضريبة على المبيعات. لتنفيذ هذا التمرين، نتبع الخطوات التالية :

١. نفتح مشروعا جديدا ونضيف تصنيف باسم TaxClass.

٢. نضيف الكود التالى إلى تصنيف TaxClass.

```
Public Class TaxClass
  Overloads Function TaxAmount (ByVal decPrice As Decimal, _
      ByVal TaxRate As Single) As String
    TaxAmount = "Price is a Decimal. Tax is $" & _
      (CStr (decPrice * TaxRate))
  End Function

  Overloads Function TaxAmount (ByVal strPrice As String, _
      ByVal TaxRate As Single) As String
    TaxAmount = "Price is a String. Tax is $" & _
      CStr ((CDec (strPrice) * TaxRate))
  End Function
End Class
```

٣. نضيف الإجراء التالى إلى تصنيف النموذج.

```
Sub ShowTax ()
  Const TaxRate As Single = 0.08
  Dim strPrice As String = "64.00"
  Dim decPrice As Decimal = 64
  Dim aclass As New taxclass ()
  MessageBox.Show (aclass.TaxAmount (strPrice, TaxRate))
  MessageBox.Show (aclass.TaxAmount (decPrice, TaxRate))
End Sub
```

٤. نضيف متحكم Button إلى النموذج ونستدعى إجراء ShowTax من داخل إجراء معالجة حدث Button1_Click.

٥. نضغط على مفتاح F5 لتشغيل التطبيق ثم ننقر على الزر الموجود على النموذج لاختبار إجراء ShowTax.

يقوم مترجم الكود فى وقت التشغيل باختيار الدالة المحملة المناسبة التى تتوافق مع المعاملات المستخدمة. وعند النقر على متحكم Button على وجه النموذج، يتم استدعاء الدالة المحملة فى البداية باستخدام معامل Price من نوع String وعرض رسالة توضح نوع بيانات Price وقيمة الضريبة. بعد النقر على زر OK فى الرسالة يقوم التطبيق باستدعاء وسيلة TaxAmount مرة أخرى واستخدام قيمة عشرية ثم عرض رسالة تفيد ذلك.

الهيمنة على الخصائص والوسائل (Overriding)

تقوم التصنيفات المشتقة بوراثة الخصائص والوسائل المعرفة فى التصنيف الأصلى. ويعتبر ذلك مفيدا لأنة يمكننا من إعادة استخدام هذه البنود عند الحاجة إليها فى التصنيف الذى نستخدمه. وإذا لم نستطع استخدام العضو الموروث كما هو ، فإن لدينا خيار استخدام كلمة Overrides لتعريف تنفيذ جديد له ، على افتراض أن الخاصية أو الوسيلة فى التصنيف الأصلى تستخدم الكلمة المفتاحية Overridable. ويمكن أيضا تظليل (Shadowing) العضو الموروث عن طريق إعادة تعريفة فى التصنيف المشتق.

فى الواقع ، يجرى استخدام الهيمنة على الأعضاء الموروثة للاستفادة من مفهوم تعدد الأوجه (Polymorphism). وتنطبق القواعد التالية على عملية الهيمنة على الوسائل.

- يمكن فقط الهيمنة على الأعضاء التى تستخدم Overridable فى التصنيف الأصلى.
- يجب أن تحتوى الوسائل التى يتم الهيمنة عليها على نفس المعاملات الموجودة فى الأعضاء الموروثة من التصنيف الأصلى.
- يمكن أن يقوم التنفيذ الجديد لوسيلة استدعاء التنفيذ الأساسى فى التصنيف الأصلى عن طريق استخدام كلمة MyBase قبل اسم الوسيلة.

لإيضاح عملية الهيمنة (Overriding) ، نستخدم التمرين التالى الذى يقوم بتعريف تصنيف أساسى هو تصنيف Payroll، وتصنيف مشتق هو تصنيف BonusPayroll، الذى يهيمن على وسيلة PayEmployee الموروثة. ويقوم إجراء RunPayroll بتكوين كائن Payroll

وكائن BonusPayroll وتمريرها إلى دالة Pay، التى تقوم بتنفيذ وسيلة PayEmployee فى كل من الكائنين. لتنفيذ هذا التمرين ننفذ الخطوات التالية :

١. نكون مشروع جديد باسم TestOverriding.

٢. نضيف تصنيف باسم Payroll إلى المشروع، ثم نضيف الكود التالى إلية :

```
Public Class Payroll
  Overridable Function PayEmployee (ByVal HoursWorked As Decimal, _
      ByVal PayRate As Decimal) As Decimal
    PayEmployee = HoursWorked * PayRate
  End Function
End Class
```

٣. نضيف تصنيف أخر إلى المشروع باسم BonusPayroll ، ثم نضيف الية الكود التالى :

```
Public Class BonusPayroll
   Inherits Payroll
   Const BonusRate As Decimal = 1.45
   Overrides Function PayEmployee(ByVal HoursWorked As Decimal, _
       ByVal PayRate As Decimal) As Decimal
     PayEmployee   =   MyBase.PayEmployee(HoursWorked,   PayRate)   *
BonusRate
   End Function
End Class
```

٤. نضيف الإجراء التالى الى تصنيف النموذج الأساسى :

```
Sub RunPayroll()
    Dim PayrollItem As Payroll = New Payroll()
    Dim BonusPayrollItem As New BonusPayroll()
    Dim HoursWorked As Decimal = 40

    MessageBox.Show("Normal pay is: " & _
    PayrollItem.PayEmployee(HoursWorked, PayRate))
    MessageBox.Show("Pay with bonus is: " & _
    BonusPayrollItem.PayEmployee(HoursWorked, PayRate))
  End Sub
```

٥. نعلن عن متغير PayRate فى بداية تصنيف النموذج الأساسى باستخدام الكود التالى :

٦. Const PayRate As Decimal = 14.75

٧. نضيف متحكم Button إلى النموذج ثم نستخدم إجراء معالجة Button1_Click فى

استدعاء إجراء RunPayroll.

٨. نضغط على مفتاح F5 لتشغيل التطبيق.

٩. ويمكن استخدام ترخيص NotOverridable و ترخيص MustOverride للتحكم في الهيمنة على الخصائص والوسائل في التصنيفات المشتقة. تستخدم NotOverridable لتعريف وسيلة في التصنيف الأصلى لايمكن الهيمنة عليها في التصنيف المشتق. وتعتبر كل وسائل التصنيف الأساسى.غير قابلة للهيمنة في تصنيف مشتق، إلا إذا تم استخدام Overridable معها. وكل وسائل التصنيف الأصلى التى تستخدم MustOverride ليس لها كود تنفيذي في التصنيف الأصلى ويجب تنفيذها في التصنيفات المشتقة. ويجب تعليم الوسائل التى تستخدم MustOverride باستخدام MustInherit.

الأحداث (Events)

يمكن تعريف الحدث بأنة رسالة مرسلة بواسطة أحد الكائنات ليعلن أن هناك شيئا ما قد حدث. وتقود الأحداث تنفيذ معظم البرامج، بمعنى أن تسلسل التنفيذ في البرنامج يتقرر على أساس أحداث تقع خارج هذا البرنامج. إن الحدث هو إشارة تشعر التطبيق بأن شيئا مهما قد حدث. على سبيل المثال، عندما يقوم المستخدم بالنقر على متحكم على سطح النموذج، يقوم النموذج بإطلاق حدث النقر (Click) واستدعاء إجراء لمعالجة هذا الحدث. كما تسمح الأحداث أيضا للمهام بالاتصال معا. من أمثلة ذلك، إذا كان التطبيق يقوم بتنفيذ عملية فرز منفصلة عن التطبيق الأساسى ثم قام المستخدم بإلغاء عملية الفرز، فإن التطبيق يقوم بإرسال رسالة إلغاء العملية إلى الإجراء الذى يقوم بعملية الفرز لإيقافها. يرتبط بالأحداث بعض المصطلحات والمفاهيم التى يمكن إيضاحها فيما يلى :

الإعلان عن الأحداث (Declaring Events)

يتم الإعلان عن الحدث في داخل التصنيفات (Classes)، الهياكل (Structures)، الوحدات (Modules)، وواجهات التصنيفات (Interfaces) باستخدام الكلمة المفتاحية Event، كما يبينها الكود التالى:

```
Event AnEvent(ByVal EventNumber As Integer)
```

إذاعة الأحداث (Raising Events)

يماثل الحدث رسالة تعلن أن شيئا مهما قد حدث. ويطلق على عملية إذاعة الحدث Raising. ويجرى إذاعة الأحداث فى Visual Basic .NET باستخدام عبارة RaiseEvent ، كما يوضحها الكود التالى :

```
RaiseEvent AnEvent (EventNumber)
```

ويجب إذاعة الأحداث فى وحدة الكود التى يتم إعلانها فيه. من أمثلة ذلك، أن التصنيف المشتق لايمكن أن يذيع حدثا موروثا من تصنيف أصلى.

مرسل الحدث (Event Sender)

يعتبر أى كائن قادرا على إذاعة حدث مرسلا لهذا الحدث، كما يعرف أيضا بمصدر الحدث. من أمثل مصادر الأحداث، النماذج (Forms)، أدوات التحكم (Controls)، والكائنات التى يكونها المستخدم.

معالج الحدث (Event Handler)

معالج الحدث هو إجراء يجرى استدعاؤه عند وقوع الحدث المقابل. وتستخدم البرامج الفرعية (Subroutines) لمعالجة الأحداث. ولا يمكن استخدام دالة (Function) لمعالجة حدث، لأن معالج الحدث لايمكن أن يعيد قيمة إلى مصدر الحدث. ويستخدم Visual Basic طريقة ثابتة لتسمية إجراءات معالجة الأحداث تتكون من اسم مرسل الحدث متبوعا بشرطة تحت السطر (Underscore) ثم اسم الحدث. على سبيل المثال اسم معالج حدث النقر على الزر Button1 هو Button1_Click.

ربط الأحداث مع إجراءات معالجة الأحداث

قبل استخدام إجراء معالجة حدث ، يجب أولا الربط بين إجراء المعالجة وبين الحدث باستخدام عبارة Handles

أو عبارة AddHandler. كما يجب الإعلان عن الكائن الذى تصدر عنة الأحداث باستخدام كلمة WithEvents عند استخدام إجراء معالجة يحتوى على فقرة Handles تحدد اسم هذا الحدث. ومع أن فقرة Handles هى الوسيلة الرئيسية لربط حدث مع إجراء معالجة، فإنها مقصورة على الربط بين الأحداث وبين إجراءات المعالجة وقت ترجمة البرنامج. وتعتبر عبارات AddHandler و RemoveHandler أكثر مرونة من فقرة Handles، لأنها تسمح لنا بربط أو إلغاء ربط الأحداث مع واحد أو أكثر من إجراءات معالجة الأحداث

فى وقت التشغيل، كما أنها لا تتطلب الإعلان عن الكائنات التى تصدر عنها الأحداث باستخدام فقرة WithEvents.

وفى بعض الحالات، مثل حالات الأحداث المرتبطة مع نماذج أو أدوات تحكم، يقوم Visual Basic .NET بتكوين إجراء معالجة خالي وربطة مع أحد الأحداث. على سبيل المثال، عند النقر المزدوج على زر أمر فى نموذج أثناء التصميم، يقوم Visual Basic.NET بتكوين إجراء معالجة وكائن لزر الأمر يحتوى على فقرة WithEvents، كما يتضح من الكود التالى:

```
Friend WithEvents Button1 As System.Windows.Forms.Button
Protected Sub Button1_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles Button1.Click
End Sub
```

إضافة أحداث إلى التصنيفات

نستخدم عبارة Event فى قسم Declarations بوحدة الكود التى يتم فيها تعريف التصنيف مع ذكر المعاملات التى يستخدمها الحدث. كما يوضحها المثال التالى :

```
Public Event PercentDone (ByVal Percent As Single,  ByRef Cancel As Boolean)
```

ولا يمكن للحدث أن يعيد قيمة، أن يكون به معاملات اختيارية، أو معامل ParamArray. ويعنى إضافة حدث إلى تصنيف، أن كائنات هذا التصنيف يمكنها إذاعة هذا الحدث. ولإذاعة الحدث، يجب استخدام عبارة RaiseEvent. ويمكننا استخدام كلمة Handles أو عبارة AddHandler لربط الحدث مع إجراء معالجة.

كتابة إجراءات معالجة الأحداث

تعتمد الطريق المستخدمة فى تكوين إجراء معالجة الحدث على الطريقة المستخدمة لربطة بالحدث. والطريقة القياسية لتكوين معالج حدث هى كلمة Handles مع كلمة WithEvents. كما يوفر لنا Visual Basic.NET طريقة أخرى لمعالجة الأحداث هى عبارة AddHandler. حيث تسمح لنا عبارة AddHandler و عبارة RemoveHandler ببدء وإيقاف معالجة الحدث ديناميكيا. ويمكن استخدام أى من المدخلين، ولكن لايجب استخدام WithEvents و AddHandler مع نفس الحدث.

معالجة الأحداث باستخدام WithEvents

تسمح لنا كلمة WithEvents بتكوين متغيرات على مستوى التصنيف أو وحدة الكود،

يمكن استخدامها مع فقرة Handles فى إجراءات معالجة الأحداث. الخطوات التالية توضح كيفية استخدام فقرة WihtEvents مع فقرة Handles لمعالجة الأحداث.

١. فى قسم Declarations بوحدة الكود التى سوف تقوم بمعالجة الحدث، نستخدم كلمة WithEvents للإعلان عن متغير خاص بالكائن مصدر الحدث. ولكى يقبل Visual Basic هذا الإعلان، يجب الإعلان أولا عن حدث فى وحدة الكود مصدر الحدث. يوضح المثال التالى كود الإعلان عن هذا المتغير:

```
Public WithEvents ClassInst As Class1
```

٢. فى محرر الكود، نختار المتغير الذى يحتوى على فقرة WithEvents الذى تم الإعلان عنة فى الخطوة السابقة ، من قائمة Class Name المنسدلة على اليسار.

٣. نختار الحدث الذى نريد معالجته من قائمة Method Name المنسدلة على الجانب الأيمن من مربع الكود. يترتب على ذلك قيام محرر الكود بتكوين إجراء معالجة خالي للحدث مع فقرة Handles تحتوى على اسم الحدث. وبدلا من ذلك، يمكن تكوين معالج الحدث يدويا طالما أنة برنامج فرعى وليس دالة يحتوى على جميع المعاملات التى يتطلبها الحدث.

٤. نضيف الكود اللازم لمعالجة الحدث إلى إجراء المعالجة باستخدام المعاملات التى يتم تمريرها. وفيما يلى مثال على ذلك :

```
Public Sub ClassInst_AnEvent (ByVal EventNumber As System.Integer) _
    Handles ClassInst.AnEvent
  MessageBox.Show ("Received event number: " & CStr(EventNumber))
End Sub
```

معالجة الأحداث باستخدام AddHandler

يمكن استخدام عبارة AddHandler لتنفيذ اتصال ديناميكي بين الأحداث وبين إجراءات معالجة الأحداث. لتنفيذ ذلك، نتبع الخطوات التالية :

١. نعلن عن متغير خاص بكائن من التصنيف مصدر الحدث الذى نريد معالجته. وعلى خلاف متغير WithEvents، يمكن أن يكون هذا المتغير محليا فى الإجراء، مثل :

```
Dim CI As New Class1 ()
```

٢. نستخدم عبارة AddHandler لتحديد اسم مرسل الحدث، وعبارة AddressOf

لتوفير اسم إجراء معالجة الحدث، على سبيل المثال :

```
AddHandler CI.AnEvent, AddressOf Ehandler
```

ويمكن استخدام أى إجراء لمعالجة الحدث طالما أنة يفى بالمعاملات التى يتطلبها الحدث.

٣. نضيف كود إلى إجراء معالجة الحدث، كما فى المثال التالى:

```
Public Sub EHandler (ByVal EventNumber As Integer)
  MessageBox.Show ("Received event number " & CStr (EventNumber))
End Sub
```

للتمرين على استخدام عبارة AddHnadler لربط حدث مع إجراء معالجة، ننفذ الخطوات التالية :

١. نفتح مشروعا جديدا ونضيف إلية تصنيف باسم Class1، ثم نضيف الكود التالى إلى التصنيف :

```
Public Class Class1
  Public Event Event1 (ByVal EventNumber As Integer)
  Sub CauseEvent (ByVal EventNumber As Integer)
    RaiseEvent Event1 (EventNumber)
  End Sub
End Class
```

٢. نضيف الإجراءات التالية فى تصنيف النموذج الأساسى:

```
Protected Sub TestEvents (ByVal EventNumber As Integer)
  Dim MyObject As New Class1 ()
  AddHandler MyObject.Event1, AddressOf Me.EventHandler
  MyObject.CauseEvent (EventNumber)
End Sub

Sub EventHandler (ByVal EventNumber As Integer)
  MessageBox.Show ("Received event number " & CStr (EventNumber))
End Sub
```

٣. نضيف متحكم Button إلى النموذج الأصلى، ونضيف الكود التالى إلى إجراء معالجة Button1_Click :

```
TestEvents (0)
```

٤. نضغط مفتاح F5 لتشغيل التطبيق، ثم ننقر على زر الأمر الموجود على النموذج. يؤدى ذلك على عرض رسالة "Received event number 0".

استخدام RemoveHandler لإيقاف معالجة الأحداث

يمكن استخدام عبارة RemoveHandler لفصل ارتباط الأحداث مع إجراءات معالجة هذة الأحداث. لتنفيذ ذلك ، نستخدم عبارة RemoveHandler لتحديد إسم مرسل الحدث، وعبارة AddressOf لتحديد اسم إجراء معالجة هذا الحدث ، كما يتضح من الكود التالى:

```
RemoveHandler CI.AnEvent, AddressOf Ehandler
```

معالجة الأحداث الموروثة من تصنيف أصلى

يمكن أن تقوم التصنيفات المشتقة بمعالجة الأحداث التى يذيعها التصنيف الأصلى عن طريق استخدام عبارة Handles MyBase <event name>. للقيام بذلك ، نعلن عن إجراء معالجة حدث فى التصنيف المشتق بإضافة العبارة السابقة الى سطر الإعلان عن إجراء معالجة الحدث ، كما يتضح من الكود التالى :

```
Public Class Class1
    Public Event SE(ByVal i As Integer)

End Class

Public Class Class2
    Inherits Class1
    Sub EventHandler(ByVal x As Integer) Handles MyBase.SE

    End Sub
End Class
```

كائنات التفويض (Delegates)

يمكن استخدام كائنات التفويض لاستدعاء الوسائل فى الكائنات الأخرى. ويطلق علي كائنات التفويض أحيانا مؤشرات الدوال (function pointers) لأنها تشبة المؤشرات إلى الدوال المستخدمة فى لغات البرمجة الأخرى ، مثل لغة ++C. ولكنها تختلف عن مؤشرات الدوال فى أنها نوع مرجعى يقوم على أساس System.Delegate ، وتدعم استدعاء الوسائل المشتركة (Shared Methods) ووسائل إمثلة التنصنيفات (Instance Methods). وتظهر فائدة كائنات التفويض عندما نحتاج الى وسيط بين الإجراء الذى يقوم بالاستدعاء وبين الإجراء الذى يتم استدعاؤه. على سبيل المثال ، يقوم Visual Basic .NET بتكوين كائنات تفويض لتمكيننا من الربط الديناميكى بين الأحداث وبين إجراءات معالجة هذة الأحداث باستخدام عبارة AddHandler ، حيث يقوم كائن التفويض فى وقت التشغيل

بتمرير الاستدعاء الى معالج الحدث المناسب. ومع أنة يمكننا تكوين كائنات التفويض ، إلا أنة فى معظم الحالات يقوم Visual Basic.NET بالقيام بهذة المهمة والاهتمام بالتفصيلات الخاصية بذلك.

واجهات استخدام التصنيفات (Interfaces)

تقوم واجهات الاستخدام بتحديد الخصائص ، الوسائل ، والأحداث التى يمكن للتصنيفات أن تنفذها. ومع أن إصدارات Visual Basic السابقة لم تكن تستطيع تكوين واجهات استخدام للتصنيفات ، إلا أن Visual Basic.NET يستطيع ذلك بواسطة عبارة Interface ، التى تمكننا من تكوين واجهات استخدام فى صورة كيانات منفصلة عن التصنيفات ، وتنفيذها باستخدام نسخة متقدمة من الكلمة المفتاحية Implements.

ومع أن واجهات الاستخدام تعرف مجموعة من الخصائص ، الوسائل ، والأحداث مثل التصنيفات ؛ إلا أنها ، على خلاف التصنيفات ، لاتقوم بتنفيذها. ويتم تنفيذ هذة الواجهات بواسطة التصنيفات على أنها كائنات منفصلة عن هذة التصنيفات. وتمثل واجهة الاستخدام عقدا يفرض قيام تصنيف بتنفيذ واجهة الاستخدام ومحتوياتها كما تم تعريفها بالضبط. بناءا على هذا العقد ، لايمكن تغيير واجهة الاستخدام بمجرد صدورها ، كما لايمكن تغيير تنفيذها.

تعريف واجهة استخدام التصنيف

يتم تعريف واجهة الاستخدام بين عبارة Interface وعبارة End Interface. بعد عبارة Interface ، يمكننا إضافة عبارة Inherits اختيارية تعرض واجهة استخدام أو أكثر يمكن وراثتها. ويجب أن تسبق عبارة Inherits كل العبارات فى التعريف فيما عدا الملاحظات. وتشمل العبارات الأخرى فى التعريف عبارات Sub ، Event ، Function ، و Property. ولايمكن أن تحتوى واجهة الاستخدام على كود تنفيذى أو عبارات مرتبطة بالكود التنفيذى ، مثل End Sub ، End Property. والعبارات الموجودة فى واجهة الاستخدام كلها عبارات عامة (Public) فى الأصل ، ولكن يمكن أيضا إعلانها Public ، Friend ، Protected ، أو Private. وكلمات التراخيص المستخدمة هى Overloads و Default ولا يسمح بباقى كلمات التراخيص(Modifiers).

وتستخدم كلمة Implements لتحديد عضو فى تصنيف ليقوم بتنفيذ واجهة استخدام

معينة. تتطلب هذة العبارة قائمة تشتمل على أعضاء التصنيف التى يجب تنفيذها. وبصفة عامة يتم تحديد عضو واجهة استخدام واحد فى هذة القائمة ، ولكن يمكن تحديد عدد من أعضاء واجهة الاستخدام. ويتكون تحديد عضو فى واجهة استخدام من إسم واجهة الاستخدام ثم نقطة ثم اسم العضو. ويمكن أن يستخدم عضو التصنيف الذى يقوم بتنفيذ عضو واجهة الاستخدام أي اسم لة دلالة. على سبيل المثال ، يوضح الكود التالى كيفية الإعلان عن إجراء فرعى باسم Sub1 يقوم بتنفيذ وسيلة فى واجهة استخدام :

```
Sub Sub1(ByVal i As Integer) Implements interfaceclass.interface2.Sub1
```

ويجب أن تكون أنواع بيانات المعاملات والأنواع العائدة الخاصة بالعضو الذى يقوم بالتنفيذ متوافقة مع إعلان أعضاء واجهة الاستخدام. ويجب أن يقوم التصنيف الذى ينفذ واجهة استخدام بتنفيذ كل أعضائه.

يوضح المثال التالى تعريف اثنين من واجهات الاستخدام. واجهة الاستخدام الثانية ، Interface2 ، ترث واجهة الاستخدام الأولى Interface1 ، كما تعرف خاصية واحدة و اثنين من الوسائل الأخرى.

```
Interface Interface1
    Sub sub1(ByVal i As Integer)
End Interface

Interface Interface2
    Inherits Interface1
    Sub M1(ByVal y As Integer)
    ReadOnly Property Num() As Integer
End Interface
```

ويقوم المثال التالى بإيضاح تنفيذ واجهة الاستخدام Interface1 السابق تعريفها فى المثال السابق :

```
Public Class ImplementationClass1
   Implements Interface1
   Sub Sub1(ByVal i As Integer) Implements Interface1.Sub1

   End Sub
End Class
```

المثال الأخير يقوم بتنفيذ واجهة الإستخدام Interface2 ، بما فيها الوسيلة الموروثة من واجهة Interface1 :

```
Public Class ImplementationClass2
  Implements Interface2
  Dim INum As Integer = 0
  Sub sub1(ByVal i As Integer) Implements Interface2.Sub1

  End Sub
  Sub M1(ByVal x As Integer) Implements Interface2.M1

  End Sub

  ReadOnly Property Num() As Integer Implements _
    Interface2.Num
    Get
      Num = INum
    End Get
  End Property
End Class
```

للتدريب على تعريف واستخدام واجهات الاستخدام ، نقوم بتنفيذ الخطوات التالية :

١. نفتح مشروع جديد بالنقر على New في قائمة File ثم النقر على مشروع. يترتب على ذلك ظهور مربع حوار New Project.
٢. نختار Windows Application من قائمة مشروعات Visual Basic.
٣. نضيف وحدة كود جديدة إلى المشروع بالنقر على Add Module في قائمة Project.
٤. نجعل اسم الوحدة الجديدة Module1.vb ثم ننقر Open. يترتب على ذلك عرض كود الوحدة.
٥. نحدد واجهة استخدام باسم MyInterface داخل وحدة Module1 عن طريق طبع عبارة Interface MyInterface بين عبارة Module وعبارة End Module ثم الضغط على مفتاح الإدخال.
٦. نحدد خاصية ، وسيلة ، وحدث في واجهة الاستخدام بإدخال الكود التالي بين عبارة Interface وعبارة End Interface :

```
Property Prop1 () As Integer
Sub Method1 (ByVal X As Integer)
Event Event1 ()
```

لتنفيذ واجهة الاستخدام، نتبع الخطوات التالية :

١. نضيف تصنيف باسم ImplementationClass بإضافة الكود التالي إلى Module1 :

```
Class ImplementationClass
```

٢. نضيف عبارة Implements في تصنيف ImplementationClass، الذي يقوم بتحديد الواجهة التي يقوم التصنيف بتنفيذها :

٣. Implements MyInterface

٤. نضيف الكود التالي إلى تصنيف ImplementationClass لتنفيذ حدث Event1 :

```
Event Event1 () Implements MyInterface.Event1
```

٥. يحتوى هذا الكود على عبارة Event لتنفيذ حدث Evern1 ، وعبارة Implements التي تحدد اسم الواجهة وعضو الواجهة الذي يتم تنفيذه.

٦. نضيف الكود التالي إلى تصنيف ImplementationClass لتنفيذ باقي أعضاء الواجهة :

```
Private pval As Integer
Property Prop1 () As Integer Implements MyInterface.Prop1
  Get
    Return pval
  End Get
  Set (ByVal Value As Integer)
    pval = Value
  End Set
End Property

Sub Method1(ByVal X As Integer) Implements MyInterface.Method1
  MsgBox("The X parameter for Method1 is " & X)
  RaiseEvent Event1()

End Sub
```

لاختبار تنفيذ الواجهة ، نتبع الخطوات التالية :

١. ننقر بزر الماوس الأيمن على النموذج الإفتراضى فى مربع Solution Explorer ثم ننقر View Code. يترتب على ذلك ، عرض تصنيف نموذج البداية.

٢. نضيف الإجراء التالي إلى تصنيف النموذج لمعالجة الأحداث التي تقع من كائنات تصنيفImplementationClass :

```
Sub EventHandler ()
  MsgBox ("The event handler caught the event")
```

```
End Sub
```

٣. نضيف الإجراء التالي إلى تصنيف Form1 لاختبار تصنيف التنفيذ :

```
Sub Test ()
Dim T As New ImplementationClass ()
Dim I As MyInterface

        I = T
    AddHandler I.Event1, AddressOf EventHandler
     I.Prop1 = 9
    MsgBox ("Prop1 was set to " & I.Prop1)
        I.Method1 (5)
End Sub
```

يقوم إجراء TestProcedure بتكوين مثل من التصنيف الذي ينفذ MyInterface، تكوين متغير من نوع interface، يربط إجراء معالج حدث مع الحدث الذى يقع بواسطة مثل التصنيف، ضبط خاصية، و تشغيل وسيلة من خلال واجهة الاستخدام.

١. نضيف كود لاستدعاء إجراء Test من داخل إجراء Form1_Load في تصنيف النموذج الأصلي :

```
Private Sub Form1_Load (ByVal sender As System.Object, _
                ByVal e As System.EventArgs) _
                Handles MyBase.Load
  Test ()
End Sub
```

٢. نضغط مفتاح F5 لتشغيل التطبيق. سوف يتم عرض رسالة "Prop1 was set to 9". وبعد النقر على زرOK ، سوف تعرض رسالة "The X Parameter for Method1 is 5". ننقر OK لكي تظهر رسالة "The event handler caught the event".

الوراثة (Inheritance)

يدعم Visual Basic .Net عمليات الوراثة، وهى القدرة على تعريف تصنيفات تستخدم لاشتقاق تصنيفات أخرى منها. وتستخدم عبارة Inherits للإعلان عن تصنيف جديد ، يطلق علية تصنيف مشتق (Derived Class)، على أساس تصنيف قائم، يعرف بالتصنيف الأصلي (Base Class). وتقوم التصنيفات المشتقة بوراثة الخصائص، الوسائل، والإحداث من التصنيفات الأصلية. كما يمكنها توسيع هذه الأعضاء وإضافة أعضاء جديدة إلى التصنيف. ويمكن للتصنيفات المشتقة أيضا الهيمنة على الوسائل الموروثة بتكوين تنفيذ

جديد للوسيلة. وكل التصنيفات التي يكونها Visual Basic .NET قابلة للوراثة بطبيعتها.

وتسمح لنا الوراثة بكتابة وتدقيق التصنيف مرة واحدة، ثم إعادة استخدام الكود لتكوين تصنيفات جديدة مشتقة. كما تسمح لنا الوراثة باستخدام تعدد الأشكال المبنى على الوراثة (Inheritance-based Polymorphism)، الذى يعنى إمكانية تعريف تصنيفات يمكن استخدامها بالتبادل فى وقت التشغيل، ولكن مع اختلاف وظائف الوسائل والخصائص التي لها نفس الأسماء.

فيما يلي نعرض القواعد التي تحكم عملية الوراثة، وكلمات التراخيص (modifiers) التي يمكن استخدامها لتغيير الطريقة التي ترث أو تورث بها التصنيفات :

- كل التصنيفات تقبل الوراثة في الأصل إلا إذا تم استخدام كلمة NotInheritable معها. ويمكن للتصنيفات أن ترث من التصنيفات الموجودة في نفس المشروع أو من التصنيفات في وحدات تجميع أخرى (assemblies) لها مراجع بالمشروع.
- وعلى خلاف اللغات التي تسمح بالوراثة المتعددة (Multiple Inheritance) ، يسمح Visual Basic .NET بالوراثة الفردية فقط بين التصنيفات. يعنى ذلك أن التصنيفات المشتقة يكون لها تصنيف أصلي واحد. ويمكن التغلب على ذلك عن طريق تنفيذ أوجه استخدام متعددة.
- ولمنع الكشف عن البنود المقيدة في التصنيف الأصلي، يجب أن يكون نوع الوصول في التصنيف المشتق مساويا أو أكثر تقييدا من نوع الوصول في التصنيف الأصلي. على سبيل المثال، لا يستطيع التصنيف العام (Public Class) أن يرث تصنيف صديق (Friend Class) أو تصنيف خاص (Private Class). والتصنيف الصديق (Friend Class) لا يمكن أن يرث التصنيف الخاص (Private Class).

ويقدم Visual Basic.NET عدد من العبارات (Statements) وكلمات التراخيص (Modifiers) لتدعيم عمليات الوراثة، فيما يلي عرض لها :

- عبارة Inherits تقوم بتحديد التصنيف الأصلي.
- كلمة الترخيص NotInheritable تمنع المبرمجين من استخدام التصنيف ليكون تصنيفا أصليا.

- كلمة الترخيص MustInherit تحدد أن التصنيف يجب استخدامه تصنيفا أصليا فقط. وهذا النوع من التصنيفات لا يمكن استنساخها مباشرة.

الهيمنة على الخصائص والوسائل فى التصنيفات المشتقة

ترث التصنيفات المشتقة تلقائيا الوسائل من التصنيف الأصلي. وعند الحاجة إلى تغيير سلوك وسيلة موروثة عن سلوكها في التصنيف الأصلي، يمكن للتصنيف المشتق الهيمنة (Overriding) على الوسيلة الموروثة وتعريف تنفيذ جديد للوسيلة في التصنيف المشتق. وللتحكم في عملية الهيمنة، نستخدم كلمات التراخيص التالية:

- Overridable – تسمح لخاصية أو وسيلة فى تصنيف بقبول الهيمنة عليها فى تصنيف مشتق.
- Overrides – تهيمن على خاصية أو وسيلة معرفة فى التصنيف الأصلى وتقبل الهيمنة.
- NotOverridable – تمنع الهيمنة على خاصية أو وسيلة فى تصنيف مشتق. والوسائل العامة (Public Methods) لاتقبل الهيمنة بطبيعتها.
- MustOverride – تتطلب الهيمنة على خاصية أو وسيلة في تصنيف مشتق. عند استخدام هذه الكلمة، فإن تعريف الوسيلة يتكون فقط من عبارة Sub، Function، أو Property. وبالتالي لا يكون هناك عبارات الإقفال، مثل End Sub أو End Function. ويجب الإعلان عن الوسائل والخصائص والأحداث التي يجب الهيمنة (MustOverride) عليها في التصنيفات التي يجب وراثتها (MustInherit).

استخدام MyBase

نستخدم الكلمة المفتاحية MyBase لاستدعاء وسائل في التصنيف الأصلي عند الهيمنة على هذه الوسائل في التصنيف المشتق. على سبيل المثال، نفترض أننا نقوم بتصميم تصنيف مشتق يقوم بالهيمنة على وسيلة موروثة من التصنيف الأصلي. يمكن للوسيلة المهيمن عليها استدعاء الوسيلة الأصلية الموجودة في التصنيف الأصلي كما يتضح من الكود التالي :

```
Class DerivedClass
  Inherits BaseClass
  Public Overrides Function CalculateShipping (ByVal Dist As Double, _
        ByVal Rate As Double) As Double
   Return MyBase.CalculateShipping (Dist, Rate) * 2
```

```
  End Function
End Class
```

ويخضع استخدام كلمة MyBase لبعض القيود التي تتضح في النقاط التالية :

- تشير كلمة MyBase إلى التصنيف الأصلي المباشر وإلى أعضائه الموروثة. ولهذا لايمكن استخدامها للوصول الى الأعضاء الخاصة (Private Members) فى ذلك التصنيف لأن الأعضاء الخاصة لا تورث.
- كلمة MyBase هى كلمة مفتاحيه وليست كائنا. ولهذا لايمكن تخصيصها لمتغير ، تمريرها الى إجراء ، أو مقارنتها باستخدام Is.
- ولا تستلزم الوسيلة التى تؤهلها كلمة MyBase، التعريف فى التصنيف الأصلي المباشر. بل يمكن تعريفها فى تصنيف أصلى موروث بطريقة غير مباشرة.
- لايمكن استخدام كلمة MyBase لاستدعاء وسيلة فى تصنيف أصلى يستخدم MustOverride.
- لايمكن استخدام MyBase لتأهيل نفسها، كما فى الصورة التالية :

```
MyBase.MyBase.Test ( )
```

استخدام الوراثة فى تكوين التصنيفات المشتقة

تؤدى عبارة Inherits إلى وراثة تصنيف كل الأعضاء غير الخاصة الموجودة فى تصنيف معين. لتنفيذ ذلك، نجعل عبارة Inherits مع اسم التصنيف الذى نريد الوراثة منة، أول عبارة فى التصنيف المشتق. وفيما عدا عبارات الملاحظات، يجب أن تأتى هذة العبارة مباشرة بعد عبارة Class.

لتوضيح عمليات الوراثة، نستخدم الكود التالى الذى يعرف تصنيفين. التصنيف الأول، هو تصنيف أصلى بة اثنان من الوسائل. والتصنيف الثانى يرث كلا الوسيلتين من التصنيف الأصلى، يهيمن على الوسيلة الثانية، ويعرف حقل باسم iField.

```
Class Class1
  Sub Method1()
    MessageBox.Show("This is a method in the base class.")
  End Sub
  Overridable Sub Method2()
    MessageBox.Show("This is another method in the base class.")
  End Sub
End Class
```

```
Class Class2
  Inherits Class1
  Public iField as Integer
  Overrides Sub Method2()
    Messagebox.Show("This is a method in a derived class.")
  End Sub
End Class
Protected Sub TestInheritance()
  Dim C1 As New class1()
  Dim C2 As New class2()
  C1.Method1()
  C1.Method2()
  C2.Method1()
  C2.Method2()
End Sub
```

يترتب على تشغيل إجراء TestInheritance، الحصول على عدد من الرسائل على الترتيب التالى :

```
"This is a method in the base class."
"This is another method in the base class."
"This is a method in the base class."
"This is a method in a derived class."
```

تعدد الاشكال (Polymorphism)

يشير تعدد الأشكال إلى المقدرة على تعريف تصنيفات متعددة تحتوى على وسائل وخصائص متشابهة الأسماء ومختلفة الوظائف يمكن استخدامها بالتبادل بواسطة كود عميل للتصنيف فى وقت التشغيل. لقد كان تقليديا إنجاز تعدد الأشكال فى Visual Basic باستخدام واجهات الاستخدام ، ويمكن الاستمرار فى استخدامها لهذا الغرض حتى الآن. غير أن Visual Basic .NET تقدم الآن خيارًا أخرًا هو استخدام الوراثة (Inheritance) أيضا لتقديم تعدد الأشكال.

تعدد الأشكال المبنى على الوراثة

معظم أنظمة البرمجة باستخدام الكائنات توفر تعدد الأشكال من خلال الوراثة. يشمل تعدد الأشكال المبنى على الوراثة، تعريف وسائل فى التصنيف الأصلي ثم الهيمنة عليها باستخدام تنفيذ جديد لها فى التصنيفات المشتقة. على سبيل المثال، يمكن تعريف تصنيف باسم BaseTax يحتوى على وسيلة لحساب ضرائب المبيعات بصفة عامة. ويمكن اشتقاق

تصنيفات من هذا التصنيف الأصلى تحتوى على وسيلة معدلة تهيمن على الوسيلة المعرفة فى التصنيف الأصلى وتقوم بحساب ضرائب المبيعات فى حالات خاصة. يحدث تعدد الأشكال هنا من حقيقة أننا نستطيع استدعاء وسيلة حساب الضريبة باستخدام كائن أي تصنيف مشتق من التصنيف الأصلي، بدون معرفة التصنيف الذى يتبعة الكائن.

إجراء TestPoly فى المثال التالى يوضح تعدد الأشكال المبنى على الوراثة :

```
Const StateRate As Double = 0.053 ' %5.3 State tax
Const CityRate As Double = 0.028  ' %2.8 City tax
Public Class BaseTax
  Overridable Function CalculateTax(ByVal Amount As Double) As Double
    Return Amount * StateRate
  End Function
End Class

Public Class CityTax
    Inherits BaseTax
  Private BaseAmount As Double
  Overrides Function CalculateTax(ByVal Amount As Double) As Double
    BaseAmount = MyBase.CalculateTax(Amount)
    Return CityRate * (BaseAmount + Amount) + BaseAmount
  End Function
End Class

Sub TestPoly()
   Dim Item1 As New BaseTax()
   Dim Item2 As New CityTax()
   ShowTax(Item1, 22.74) ' $22.74 normal purchase.
   ShowTax(Item2, 22.74) ' $22.74 city purchase.
 End Sub

Sub ShowTax(ByVal Item As BaseTax, ByVal SaleAmount As Double)
   Dim TaxAmount As Double
   TaxAmount = Item.CalculateTax(SaleAmount)
   MsgBox("The tax is: " & Format(TaxAmount, "C"))
End Sub
```

فى المثال السابق، يتم تمرير كائن باسم Item من نوع BaseTax الى إجراء ShowTax. ويمكن أيضا تمرير كائنات من أنواع التصنيفات المشتقة من تصنيف BaseTax. كما يمكن أيضا إضافة تصنيفات جديدة مشتقة من تصنيف BaseTax بدون تغيير الكود الموجود

بإجراء ShowTax.

تعدد الأشكال المبنى على واجهات استخدام التصنيفات

تقدم واجهات الاستخدام (Interfaces) طريقة أخرى لإنجاز مفهوم تعدد الأشكال فى Visual Basic.NET. وكما سبق إيضاحه ، تقوم واجهات الاستخدام بالإعلان عن الخصائص والوسائل ولكنها لاتقوم بتنفيذها. لتحقيق تعدد الأشكال ، نكون واجهة استخدام ونقوم بتنفيذها بطرق مختلفة فى تصنيفات متعددة. يمكن بعد ذلك استدعاء الوسيلة المنفذة فى أي كائن بنفس الطريقة.

المثال التالى يعرف واجهة استخدام باسم Shape2 ، ويتم تنفيذها فى تصنيف باسم RightTriangleClass2 وفى تصنيف RectangleClass2. ويقوم إجراء يسمى ProcessShape2 باستدعاء وسيلة CalculateArea فى أمثلة تصنيف RightTriangleClass2 أو تصنيف RectangleClass2 :

```
Sub TestInterface()
  Dim RectangleObject2 As New RectangleClass2()
  Dim RightTriangleObject2 As New RightTriangleClass2()
  ProcessShape2(RightTriangleObject2, 3, 14)
  ProcessShape2(RectangleObject2, 3, 5)
End Sub

Sub ProcessShape2(ByVal Shape2 As Shape2, ByVal X As Double, _
    ByVal Y As Double)
  MessageBox.Show ("The area of the object is " _
    & Shape2.CalculateArea(X, Y))
End Sub

Public Interface Shape2
  Function CalculateArea(ByVal X As Double, ByVal Y As Double) As Double
End Interface

Public Class RightTriangleClass2
  Implements Shape2
  Function CalculateArea(ByVal X As Double, _
      ByVal Y As Double) As Double Implements Shape2.CalculateArea
     Return 0.5 * (X * Y)
  End Function
End Class
```

```
Public Class RectangleClass2
  Implements Shape2
  Function CalculateArea(ByVal X As Double, _
      ByVal Y As Double) As Double Implements Shape2.CalculateArea
    Return X * Y
  End Function
End Class
```

مصادر واستخدامات الكائنات (Sources and Uses of Objects)

تأتى الكائنات من مصادر متعددة، منها الكائنات التى يقدمها Visual Basic.NET، ومنها الكائنات التى تقدمها التطبيقات الأخرى. ويتم استخدام الكائنات من خلال استخدام الخصائص والوسائل التى تحتوى عليها. وبجانب استخدام الكائنات التى تكونها التطبيقات الأخرى، يمكننا تكوين كائنات فى تطبيقاتنا. يناقش هذا القسم من أين تأتى الكائنات وكيفية استخدامها.

الكائنات الداخلية (Internal Objects)

الكائنات الداخلية (Built-in Objects) هى الكائنات التى يوفرها Visual Basic.NET داخليا. تشمل تلك الكائنات الأنواع العددية الأولية ، مثل الرقم الصحيح (Integer) والرقم ذات الدقة المضاعفة (Double) بالإضافة الى المصفوفات والسلاسل، وهى كائنات لا تحتاج الى تكوين مراجع لها قبل استخدامها فى تطبيقاتنا. ومن الكائنات الداخلية أيضا أمثلة التصنيفات فى المشروع الجارى العمل بة. ويمكن استخدام هذة التصنيفات داخل مشروعاتنا، كما يمكن جعلها متاحة للاستخدام بواسطة التطبيقات الأخرى بوضعها فى وحدات التجميع (assemblies) التى تتكون منها التطبيقات.

الكائنات الخارجية (External Objects)

الكائنات الخارجية هى تلك الكائنات غير المتاحة للاستخدام فى التطبيق افتراضيا ، ولكنها تأتى من مشروعات أخرى أو وحدات تجميع (assemblies). لاستخدام هذا النوع من الكائنات ، يجب تكوين مراجع لها فى المشروعات التى نقوم بتكوينها. والمصدر الشائع للحصول على الكائنات الخارجية فى Visual Basic.NET هو وحدات التجميع (assemblies) التى يشتمل عليها نظام NET Framework. وهناك بعض الكائنات فى نظام NET Framework يجرى معاملتها على أنها كائنات داخلية ، ولكن معظم وحدات التجميع الموجودة بهذا النظام يجب استيرادها داخل التطبيقات باستخدام عبارة Imports قبل أن

نستطيع استخدامها. ويمكن تعريف وحدات التجميع (assemblies) بأنها وحدات بناء نظام NET Framework. وتأخذ هذة الوحدات شكل ملفات تنفيذية (exe) أو ملفات مكتبات ربط ديناميكى (dll). وتعتبر وحدة التجميع هى الوحدة الأساسية للنشر ، التحكم فى الإصدارات، إعادة الاستخدام، وتصاريح الأمان بالنسبة للتطبيقات التى تستخدم نظام NET Framework. ويتم استخدام محتويات وحدات التجميع و تكوين مراجع لها فى Visual Basic .NET بنفس الطريقة التى نستخدم بها مكتبات الأنواع فى الإصدارات السابقة من Visual Basic. والذى يجعل وحدات التجميع مختلفة عن الملفات التنفيذية (exe files.) وملفات الربط الديناميكى (dll files.) فى الإصدارات السابقة من ويندوز، هو أنها تحتوى على كل المعلومات التى يمكن أن نجدها فى مكتبة أنواع بالإضافة الى معلومات عن كل شئ يلزم لاستخدام التطبيق أو المكون.

ضبط واسترجاع قيم الخصائص (Setting and Retrieving Properties)

يمكن ضبط خصائص النماذج وأدوات التحكم فى Visual Basic .NET أثناء التشغيل باستخدام الكود ، أو أثناء التصميم باستخدام نافذة الخصائص (Properties Window). والكائنات الأخرى ، مثل الكائنات الموجودة بوحدات التجميع (assemblies) أو الكائنات التى نقوم بتكوينها بأنفسنا ، يمكن ضبطها باستخدام الكود فقط.

ويمكن أن تقبل الخصائص القراءة فقط (Read-only Properties) ، كما يمكن أن تقبل هذة الخصائص القراءة وتغيير القيمة (Read-Write Properties). والهدف من تغيير قيمة خاصية بأحد الكائنات هو تغيير شكل ظهور أو سلوك هذا الكائن. على سبيل المثال ، نقوم بتغيير خاصية Test فى مربع نص (Test Box) لكى نغير محتويات المربع.

والهدف من قراءة قيمة خاصية بأحد الكائنات ، هو معرفة حالة الكائن قبل تنفيذ أعمال أخرى على هذا الكائن بواسطة الكود الذى نقوم بكتابتة. على سبيل المثال، يمكننا استعادة خاصية Text بمربع نص لمعرفة محتوياتة قبل تنفيذ بعض الكود الذى قد يغير هذا المحتوى.

ولضبط قيم خاصية ، نستخدم صيغة الكود التالية :

```
object.property = expression
```

فيما يلى نموذج من الكود الذى يقوم بضبط بعض خصائص مربع نص ، وهى خاصية

قمة مربع النص (Top Property)، خاصية ظهور أو إخفاء مربع النص(Visible Property)، وخاصية النص (Text Property).

```
TextBox1.Top = 200
TextBox1.Visible = True
TextBox1.Text = "hello"
```

وللحصول على قيمة خاصية ، نتبع صيغة الكود التالية :

```
variable = object.property
```

ويمكن تغيير قيمة الخاصية بطريقة أخرى ، كما يتضح من الكود التالى الذى يقوم بتغيير قيمة خاصية Top فى مربع النص السابق :

```
TextBox1.Top += 20
```

استخدام الوسائل (Using Methods)

يمكن تعريف الوسائل بأنها إجراءات مرتبطة بالكائنات. وتمثل الوسائل أفعال يمكن أن يقوم بها الكائن الذى توجد بة الوسيلة، على عكس الخصائص والحقول التى تمثل معلومات يحتفظ بها الكائن. ويمكن أن تؤثر الوسيلة عند تنفيذها على القيم الموجودة بالخصائص والحقول. على سبيل المثال ، يمكن تغيير محتويات قائمة سرد عن طريق استخدام وسيلة Clear لحذف هذة المحتويات ووسيلة Add لإضافة بنود الى القائمة.

ويمكن أن تستخدم الوسائل معاملات ، كما يمكن أن تعيد قيمة بعد التنفيذ. وبصفة عامة يجرى استخدام الوسائل مثل استخدام الإجراءات الفرعية (Subroutines) ، الدوال (Functions) ، فيما عدا أننا نقوم بتأهيل الوسيلة باسم الكائن الذى تخصة الوسيلة. وعندما لانقوم بتأهيل الوسيلة ، يجرى استخدام المتغير Me لتأهيلها.

لاستخدام وسيلة لاتتطلب معاملات ، نستخدم الصيغة التالية :

```
object.method()
```

فى المثال التالى ، تقوم وسيلة Hide بإخفاء الزر Button1 :

```
Button1.Hide()
```

ولاستخدام وسيلة تتطلب معاملات متعددة ، نضع المعاملات بين أقواس ونفصل بينها بعلامة الفاصلة. المثال التالى يوضح استخدام وسيلة MsgBox التى تتطلب معاملات تحدد الرسالة التى تعرض ونمط مربع الرسالة :

```
MsgBox("Database update complete", _
     MsgBoxStyle.OKOnly Or MsgBoxStyle.Exclamation, _
     "My Application")
```

لاستخدام وسيلة تعيد قيمة، نخصص القيمة العائدة لأحد المتغيرات، أو نستخدم استدعاء الوسيلة فى صورة معامل عند استدعاء وسيلة أخرى. الكود التالى يقوم بتخزين القيمة العائدة عند تنفيذ وسيلة MsgBox، فى متغير:

```
Dim Response As MsgBoxResult
Response = MsgBox("Do you want to exit?", _
           MessageBoxButtons.YesNo Or MsgBoxStyle.Question, _
            "My Application")
```

ويقوم المثال التالى باستخدام قيمة تمثل طول سلسلة عائدة من تنفيذ وسيلة Len ، معاملا فى وسيلة MsgBox:

```
Dim MyStr As String = "Some String"
MsgBox("String length is : " & Len(MyStr))
```

تنفيذ عدد من الاعمال باستخدام نفس الكائن

نحتاج فى الغالب، إلى تنفيذ عدد من الأعمال على نفس الكائن. قد تشمل هذه الأعمال ضبط قيم بعض خصائص الكائن وتنفيذ بعض الوسائل التى يحتوى عليها. يمكن تنفيذ ذلك بكتابة عبارات عديدة تستخدم نفس الكائن وتقوم كل منها بتنفيذ مهمة محددة. ويمكن القيام بذلك بطريقة أسهل عن طريق استخدام عبارة With...End With، كما يتضح من المثال التالى :

```
Private Sub UpdateForm1( )
With Button1
    .Text = "OK"
    .Visible = True
    .Top = 24
    .Left = 100
    .Enabled = True
    .Refresh ()
End With
End Sub
```

ويمكن أيضا استخدام عبارات With...End With المتداخلة عن طريق وضع إحدى العبارات داخل عبارة أخرى، كما فى المثال التالى الذى يقوم بضبط قيم عدد من خصائص نموذج AnotherForm فى قائمة With الخارجية، وضبط عدد من خصائص مربع النص

TextBox1 فى قائمة With الداخلية :

```
Sub SetupForm()
Dim AnotherForm As New Form1()
With AnotherForm
        .Show()   ' Show the new form.
        .Top = 250
        .Left = 250
        .ForeColor = Color.LightBlue
        .BackColor = Color.DarkBlue
        With AnotherForm.Textbox1
            .BackColor = Color.Thistle
            .Text = "Some Text"
        End With
End With
End Sub
```

يجب ملاحظة أن الكود الموجود بداخل عبارة With المتداخلة يشير الى الكائن المتداخل ولايؤثر على الخصائص المذكورة فى عبارة With الخارجية.

كائنات النماذج

تستخدم التطبيقات النماذج لتكوين واجهات الاستخدام. والنموذج الذى يتم عرضة أمام المستخدم هو كائن مستنسخ من تصنيف يحدد كيفية عرض النماذج وتحديد وظائفها. ويمكن إضافة خصائص ووسائل خاصة بنا الى أحد النماذج والوصول إليها من نماذج وتصنيفات أخرى داخل التطبيق.

إضافة وسيلة جديدة فى نموذج

يمكن إضافة وسيلة إلى نموذج باستخدام الكود التالى :

```
Public Sub PrintMyJob()
End Sub
```

إضافة حقل جديد إلى نموذج

يمكن الإعلان عن متغير عام (Public Variable) فى وحدة النموذج باستخدام الكود التالى :

```
Public IDNumber As Integer
```

للوصول إلى وسائل فى نموذج أخر

١. نكون مثل جديد من النموذج الذى نريد الوصول إلى الوسائل الموجودة بة. وعند

استخدام اسم نموذج، فإننا فى الحقيقة نستخدم التصنيف الذى يتبعة النموذج.

٢. نخصص النموذج لمتغير من نوع كائن (Object Variable). يحتوى هذا المتغير على مرجع للنموذج السابق تكوينة. يوضح المثال التالى هاتين الخطوتين فى خطوة واحدة ثم يقوم باستدعاء وسيلة PrintMyJob فى النموذج الجديد.

```
Dim newForm1 As New Form1
newForm1.PrintMyJob
```

ويجب ملاحظة أن النموذج الجديد فى المثال السابق، لايتم عرضة. لأنة ليس من الضرورى عرض أحد النماذج لكى نتمكن من استخدام وسائلة. ويمكننا عرض النموذج الجديد باستخدام الكود التالى :

```
newForm1.show
```

من الملاحظ فى المثال السابق أيضا، استخدام كلمة New لتكوين كائن من تصنيف Form1. يرجع ذلك الى أن الكائنات يجب تكوينها قبل استخدامها. لتوضيح ذلك، نعرض الكود التالى :

```
Dim Button1 As System.Windows.Forms.Button()
Dim Button2 As New System.Windows.Forms.Button()
```

العبارة الأولى فى الكود السابق تعلن عن المتغير Button1 من نوع كائن Button. يحتوى هذا المتغير على قيمة Nothing التى تمثل أحد القيم المعرفة فى النظام، إلى أن يتم تخصيص كائن من نوع Button لة. بينما تقوم العبارة الثانية بالإعلان عن المتغير Button2 من نوع Button ويمكن أن يحتوى على مرجع كائن من نوع Button، كما تقوم باستخدام الكلمة المفتاحية New لتكوين كائن Button وتخصيصة للمتغير Button2.

وبالنظر إلى أن النماذج والكائنات هى فى الحقيقة تصنيفات ، لذا يمكننا استخدام كلمة New لتكوين كائنات جديدة من هذة البنود عند الحاجة. لتوضيح ذلك، نتبع الخطوات التالية :

١. نفتح مشروع جديد ونضع متحكم Button وعدد أخر من أدوات التحكم على سطح نموذج يسمىForm1.

٢. نضيف الكود التالى الى إجراء معالجة حدث Click بكائن Button:

```
Dim f AS New Form1
f.Show
```

٣. نقوم بتشغيل التطبيق، وننقر على الزر الموجود على سطح النموذج عدة مرات لتكوين عدة نسخ من النموذج.

٤. نحرك النموذج الأمامي جانبا لرؤية النماذج الأخرى التى تم تكوينها بالنقر على الزر فى الخطوة السابقة.

كما يمكن استخدام كلمة New لتكوين كائنات من التصنيفات التى نقوم بتكوينها:

١. نفتح مشروع جديد، ونضع متحكم Button على نموذج يسمى Form1.

٢. من قائمة Project، نختار Add Class لإضافة تصنيف الى المشروع.

٣. نجعل اسم التصنيف الجديد TestNew.vb.

٤. نضيف إجراء ShowFrm التالى إلى تصنيف TestNew :

```
Public Class TestNew
  Sub ShowFrm()
Dim frmNew As Form1
frmNew = New Form1
frmNew.Show()
        frmNew.WindowState = 1
  End Sub
End Class
```

٥. نضيف الكود التالى إلى إجراء معالجة حدث Button1_Click الخاص بالزر الذى على سطح النموذج:

```
Protected Sub Button1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button1.Click
Dim clsNew As New TestNew()
    clsNew.ShowFrm()
End Sub
```

٦. لاستخدام هذا المثال ، نقوم بتشغيل التطبيق ثم ننقر على زر الأوامر الموجود على النموذج عدة مرات. ترتب على ذلك ظهور أيقونة نموذج على شريط المهام عند تكوين كل مثل من أمثلة تصنيف TestNew.

تمرير الكائنات إلى الإجراءات (Passing Objects to Procedures)

يسمح لنا Visual Basic .NET بتمرير الكائنات من خلال المعاملات الى الإجراءات بنفس الطريقة التى نمرر بها أنواع المعاملات الأخرى. لإيضاح تمرير كائن Form إلى إجراء، نتبع

الخطوات التالية :

١. نقوم بتكوين مشروع جديد ونضيف زر أوامر باسم Button1 الى النموذج الإفتراضى.

٢. نضيف الإجراء التالى إلى تصنيف النموذج الأساسى :

```
Sub CenterForm(ByVal TheForm As Form)
  Dim RecForm As rectangle = Screen.GetBounds(TheForm)
  TheForm.Left = CInt((RecForm.Width - TheForm.Width) / 2)
  TheForm.Top = CInt((RecForm.Height - TheForm.Height) / 2)
End Sub
```

٣. ندخل الكود التالى فى إجراء معالجة حدث Button1_click:

```
Protected Sub Button1_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs)
  Dim newForm As New Form1()
  newForm.Show()
  CenterForm(newForm)
End Sub
```

٤. لتشغيل التطبيق نضغط على مفتاح F5. يؤدى ذلك الى تكوين نموذج جديد وعرضة فى منتصف الشاشة.

ويمكن أيضا تمرير مرجع كائن الى إجراء على نموذج أخر، وفى داخل الإجراء نخصص المرجع لمتغير كائن جديد. لإيضاح كيفية تنفيذ ذلك، نتبع الخطوات التالية :

١. نفتح مشروع جديد يحتوى على النموذج الإفتراضى Form1.

٢. نضيف نموذج أخر باسم Form2 الى المشروع.

٣. نضع كائن مربع صورة على كل نموذج.

٤. نجعل اسم مربع الصورة فى النموذج الأول PictureBox1، واسم مربع الصورة فى النموذج الثانى PictureBox2.

٥. نخصص صورة لمربع الصورة PictureBox2 بالنقر على خاصية Image فى نافذة Properties. يمكننا بعد ذلك اختيار صورة من دليل الويندوز.

٦. نضيف الكود التالى إلى نموذج Form2 :

```
Public Sub GetPicture(ByVal x As PictureBox)
  Dim objX As PictureBox
  objX = x
```

```
    objX.Image = PictureBox2.Image
  End Sub
```

٧. نضيف الكود التالى فى إجراء معالجة حدث Form1_Click :

```
  Protected Sub Form1_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs)
    Dim newForm2 As New Form2()
    newForm2.GetPicture(PictureBox1)
  End Sub
```

٨. نقوم بتشغيل التطبيق والنقر على نموذج Form1. يترتب على ذلك ظهور صورة النموذج Form2 فى مربع الصورة بنموذج Form1.

فى هذا المثال، يقوم إجراء معالجة حدث Form1_Click باستدعاء إجراء GetPicture فى نموذج Form2 وتمرير مربع الصورة الخالي الية. فى إجراء GetPicture بالنموذج الثانى يتم تخصيص خاصية Impage التابعة لمربع الصورة على نموذج Form2 لمربع الصورة الخالى على نموذج Form1 ، وعرض الصورة الموجودة على النموذج Form2 على النموذج Form1.

إدارة مجموعات الكائنات

هناك طريقتان لوضع الكائنات فى مجموعات: الطريقة الأولى هى تكوين مصفوفات (Arrays) من الكائنات، و الطريقة الثانية هى تكوين مجموعات (Collections) من الكائنات. المصفوفات، تعتبر هياكل غير مرنة لأننا لا نستطيع تغيير حجم المصفوفة أثناء التشغيل إلا باستخدام عبارة Redim فى كود التطبيق لإعادة الإعلان عن المصفوفة. كما أن كل أعضاء المصفوفة يجب أن تكون من نفس نوع البيانات. من ناحية أخرى، تتواجد أعضاء المصفوفة بالتسلسل ويمكن أن يكون هناك أعضاء خالية فى التسلسل. لهذة الأسباب، تعتبر المصفوفات أكثر فائدة لتكوين والعمل مع عدد ثابت من الكائنات المحددة النوع بدقة.

وأما فيما يختص بالمجموعات، فإنها توفر لنا طريقة أكثر مرونة للعمل مع الكائنات. وتعتبر المجموعة تصنيفا، ولهذا يجب الإعلان عنها قبل البدء فى استخدامها وإضافة أعضاء إليها. وتتميز المجموعة بقدرتها على التمدد أو الانكماش ديناميكيا كلما تغيرت احتياجات البرنامج. ويمكن تخصيص مفتاح لأى كائن نقوم بإضافة إلى المجموعة لكى نستطيع استخراجه ومعالجتة باستخدام هذا المفتاح.

تكوين مصفوفات من الكائنات

نقوم بالإعلان عن مصفوفة من نوع أحد الكائنات واستخدامها كما نفعل مع المصفوفات من أنواع البيانات الأخرى. ويمكن استخراج أي عضو فى المصفوفة باستخدام موقعة فى فهرس أعضاء المصفوفة ، كما يمكن معالجته كما نعالج أى كائن من نفس النوع. وتحتوى المصفوفات أيضا على وظائف داخلية بها، مثل وظائف البحث والفرز، والتى يمكن الوصول إليها من خلال متغير المصفوفة. ولإيضاح كيفية تكوين مصفوفة من الكائنات، نتبع الخطوات التالية:

١. نعلن عن المصفوفة كما هو مبين فى الكود التالى. ونظرا لأن ترتيب أعضاء المصفوفة يبدأ من الصفر، لذا يكون عدد الأعضاء بها أكثر بواحد من العدد المذكور بين القوسين عند الإعلان عنها.

```
Dim x(10) As Widget
```

٢. نقوم بتكوين كائن كل عضو فى المصفوفة باستخدام حلقة For فى الكود التالى :

```
Dim q As Integer
 For q = 0 to 10
 x(q) = New Car()
Next
```

٣. أو نضيف مراجع كائنات موجودة إلى أعضاء المصفوفة ، كما يتضح من الكود التالى:

```
Dim myWidnet As New Widget()
 x(0) = myWidget
 x(1) = myWidget
```

استخدام المجموعات في Visual Basic.NET

يمكن تعريف المجموعة (Collection) بأنها طريقة لتجميع الكائنات ذات العلاقة. ويقدم لنا Visual Basic تصنيف خاص بالمجموعة (Collection Class) لتمكيننا من تكوين مجموعات خاصة بنا. يوفر لنا هذا التصنيف وظائف داخلية تمكننا من الحركة داخل المجموعة فى حلقة باستخدام عبارات For Each...Next واستخدام أعضاء المجموعة بواسطة أرقامها فى فهرس المجموعة. ولايعنى وجود هذا التصنيف أن جميع كائنات المجموعات التى نستخدمها يتم استنساخها منة، بل أن هناك كائنات مجموعات أخرى ذات أنواع مختلفة عن الكائنات المستنسخة من تصنيف Collection المذكور. من أمثلة

كائنات المجموعات التى تعتبر من نوع مختلف عن كائنات تصنيف Collection، مجموعة أدوات التحكم الموجودة على نموذج الويندوز (Controls Collections). وتختلف أنواع هذة المجموعات فى الوسائل التى تحتوى عليها، طريقة تخزين مراجع الكائنات، استخدام قيم الفهرس، وغيرها. ولهذا، يحتوى هذا القسم على شرح لأهم الفرو قات بين أنواع المجموعات المختلفة.

- هناك مجموعات يبدأ فهرس أعضائها بالصفر وأخرى يبدأ فهرسها برقم واحد. من أمثلة المجموعات التى تبدأ بالصفر ، مجموعة Controls ، بينما يبدأ فهرس كائن من تصنيف Collection برقم واحد. وتتميز المجموعة التى تبدأ بالواحد بأن قيمة خاصية Count بها تمثل أخر رقم فى الفهرس ، بينما تكون قيمة خاصية Count فى المجموعة التى يبدأ فهرسها بالصفر أكبر من ترتيب أخر بند فى المجموعة بواحد. والفهرس القياسى للمجموعات فى نظام NET Framework يبدأ بالصفر ، بينما يبدأ فى Visual Basic بالواحد لغرض التوافق مع الإصدارات السابقة.
- هناك الكثير من المجموعات فى Visual Basic التى تسمح لنا بالوصول إلى بنودها باستخدام فهرس رقمي أو مفتاح يحتوى على سلسلة من الرموز ، كما هو الحال فى تصنيف Collection فى Visual Basic. ويمكن أيضا إضافة بنود الى كائنات Collection بدون تحديد مفتاح. على العكس من ذلك، هناك بعض أنواع المجموعات، مثل System.Collections.ArrayList، تسمح فقط باستخدام الفهرس الرقمى.
- وتختلف المجموعات أيضا فى قابليتها لإضافة بنود إليها، وفى كيفية إضافة البنود. وبالنظر إلى أن كائن Collection فى Visual Basic يعتبر أداة برمجة عامة، لذلك يعتبر أكثر مرونة عن غيرة من المجموعات. يحتوى هذا الكائن على وسيلة Add لإضافة البنود إلى المجموعة، ووسيلة Remove لحذف البنود منها. من ناحية أخرى، هناك بعض المجموعات الخاصة لا تسمح للمبرمج بإضافة وحذف البنود باستخدام الكود.

تصنيف Collection فى Visual Basic.NET

تستخدم المجموعات فى Visual Basic لتتبع الكثير من الأشياء ، مثل كل أدوات

التحكم على نموذج ، كما يمكن للمبرمج تكوين مجموعاته الخاصة لتنظيم ومعالجة الكائنات. يبين الجزء التالى من الكود كيفية استخدام وسيلة Add فى كائن مجموعة للاحتفاظ بقائمة من الكائنات التى يقوم المستخدم بتكوينها.

١. نعلن ونستنسخ كائن مجموعة widgetCollection من تصنيف Collection.

```
Public widgetCollection As New Collection
```

٢. نكون كائن Widget جديد ونضيفة إلى مجموعة widgetCollection.

```
Private Sub MakeAWidget()
  Dim tempWidget As New Widget()
  widgetCollection.Add(tempWidget)
End Sub
```

تقوم مجموعة widgetCollection فى هذا المثال، بتنظيم والكشف عن كائنات Widget التى يجرى تكوينها باستخدام وسيلة MakeAWidget. ويمكن استخراج مرجع لكل كائن Widget من خلال فهرس المجموعة. ويتم تعديل حجم المجموعة أتوماتيكيا عند إضافة كائن جديد إليها. ويمكننا استخدام عبارات For Each...Next لتكرار الدوران خلال المجموعة. وعندما نرغب فى تخصيص مفتاح لكل كائن Widget لإستخدامة في استخراج الكائن من المجموعة ، يمكننا إدخال سلسلة نص فى المعامل الثاني بوسيلة Add.

كلمة New فى الإعلان عن متغير widgetCollection تقوم بتكوين كائن جديد من تصنيف Collection. وبعد تكوين هذا الكائن، يتم وضع المرجع إلية فى المتغير widgetCollection.

ويشتمل كل كائن Collection على خصائص ووسائل يمكن استخدامها فى إدراج، حذف، واستخراج بنود المجموعة. تشمل هذة البنود ما يلى :

- وسيلة Add ، تستخدم فى إضافة بنود إلى المجموعة.
- وسيلة Remove ، تستخدم فى حذف بنود من المجموعة.
- خاصية Count ، تعيد عدد البنود فى مجموعة.
- خاصية Item ، تعيد مرجع إلى بند فى المجموعة ممثلا فى فهرس أو مفتاح.

هذة الخصائص والوسائل توفر الخدمات الأساسية التى تحتاجها المجموعات. على سبيل المثال، وسيلة Add لا تستطيع تدقيق نوع الكائن المضاف إلى المجموعة للتحقق من أن

المجموعة تحتوى على نوع واحد من الكائنات. هذه الخصائص والوسائل تعتمد على المفاتيح والفهارس فى أداء وظائفها. المفتاح يمكن تعريفة بأنة قيمة من نوع سلاسل الرموز. ويمكن أن يكون اسما أو رقما يتم تحويلة إلى سلسلة. وتمكننا وسيلة Add من الربط بين مفتاح وبين أحد البنود الذى تحتوى عليها المجموعة. والفهرس فى تصنيف Collection يكون رقما صحيحا بين الواحد وبين عدد البنود فى المجموعة. ويمكننا استخدام الفهرس لتكرار فحص البنود فى المجموعة. لتوضيح ذلك، نعرض المثال التالى الذى يفترض احتواء المتغير employeesCollection على مرجع كائن مجموعة، كما يقوم بزيادة معدل الأجر لكل موظف فى المجموعة بنسبة ١٠٪ باستخدام طريقتين.

الطريقة الأولى: استخدام حلقة For لتكرار المعالجة داخل المجموعة

```
Option Strict On
Dim counter As Integer
Dim emp As Employee
For counter = 1 To employeesCollection.Count
  emp = CType(employeesCollection(counter), Employee)
  emp.Rate *= 1.1
Next
```

الطريقة الثانية: استخدام حلقة For Each لتكرار المعالجة داخل المجموعة

```
Dim emp As Employee
For Each emp In employeesCollection
  emp.Rate *= 1.1
Next
```

ويجب ملاحظة أننا لا نستطيع استخدام عبارة For Each مع المجموعات التى تحتوى على أنواع مختلفة من الكائنات عن تلك التى نقوم بتكرار معالجتها.

إضافة بنود إلى مجموعة

نستخدم وسيلة Add لإضافة بنود إلى مجموعة، كما يتضح من الكود التالى:

```
object.Add (Item, Key)
```

على سبيل المثال، لإضافة كائن العميل custNew إلى مجموعة العملاء customerCollection وجعل مفتاح الإضافة يساوى قيمة خاصية ID بكائن العميل، نستخدم الكود التالى :

```
customerCollection.Add (custNew,  custNew.ID)
```

يفترض هذا الكود أن قيمة خاصية ID من نوع سلسلة الرموز (String). وإذا كانت

قيمة الخاصية من نوع الأرقام الصحيحة، نستخدم دالة ToString لتحويل الرقم الى سلسلة، كما يوضحة الكود التالى:

```
customerCollection.Add (custNew,  custNew.ID.ToString)
```

ويعتبر استخدام المفاتيح اختياريا عند إضافة كائنات إلى مجموعة. لأنة يمكننا إضافة كائن إلى مجموعة بدون استخدام مفتاح، كما يتضح من الكود التالى :

```
customerCollection.Add (custNew)
```

ويمكن استخدام معامل before ومعامل after للمحافظة على ترتيب الكائنات داخل المجموعة. بموجب ذلك، يتم وضع العضو الذى تجرى إضافته قبل أو بعد العضو المعرف بمعامل before أو after على الترتيب. على سبيل المثال جعل before تساوى 1، يضع البند المضاف فى بداية المجموعة. لأن مجموعة الكائنات تبدأ الترقيم من الواحد ، كما يوضحة المثال التالى :

```
customerCollection.Add(custNew,  custNew.ID,  1)
```

وبالمثل، يقوم معامل after بإضافة بند بعد رقم الفهرس المحدد. الكود التالي يقوم بإضافة بند فى الترتيب الثالث:

```
customerCollection.Add (custNew, custNew.ID,  , 2)
```

ولا يجب إضافة قيم لكل من معامل after ومعامل before فى نفس الوقت.

حذف بنود من مجموعة

نستخدم وسيلة Remove لحذف أحد البنود من مجموعة. والكود اللازم لذلك هو :

```
object.Remove (index | key)
```

معامل index فى الصيغة السابقة، يمكن أن يكون ترتيب البند الذى نريد حزفة فى المجموعة، أو مفتاح البند. على سبيل المثال، إذا كان مفتاح ترتيب البند الثالث فى مجموعة هو "C0989"، يمكننا استخدام أحد العبارتين التاليتين لحذف البند من المجموعة:

```
customerCollection.Remove(3)
customerCollection.Remove("C0989")
```

استخراج البنود من مجموعة

لاستخراج بنود معينة من مجموعة، نستخدام خاصية Item كما يوضحها التركيب اللغوى التالى :

```
variable = object.Item (index)
```

ويمكن أن يكون معامل index فى التركيب السابق موقع البند فى المجموعة أو مفتاح البند. باستخدام المثال السابق، تقوم أي من العبارتين التاليتين باستخراج العنصر الثالث فى مجموعة:

```
custCurrent = customerCollection.Item(3)
custCurrent = customerCollection.Item("C0989")
```

وتعتبر خاصية Item هى الخاصية الافتراضية فى كائن Collection، ولهذا يمكن حذفها عند الوصول إلى بند فى مجموعة. على هذا الأساس، يمكن إعادة كتابة العبارتين السابقتين بالطريقة التالية :

```
custCurrent = customerCollection (3)
custCurrent = customerCollection ("C0989")
```

ويمكن أيضا استخدام رمز التأهيل (!) للوصول إلى عضو فى مجموعة باستخدام مفتاحة بدون تأهيل هذا المفتاح بعلامات الاقتباس أو الأقواس. ولذلك ، يمكن إعادة كتابة المثال السابق كما يلي :

```
custCurrent = customerCollection ! C0989
```

الكائنات غير محددة النوع

قد نحتاج أحيانا إلى العمل مع متغيرات من نوع Object بدلا من استخدام كائنات محددة النوع. وتتميز المتغيرات من نوع Object بأنها يمكن أن تحتوى على كائنات أي تصنيف.

تحديد التصنيف الذى يتبعة الكائن وقت التشغيل

قد نحتاج إلى تحديد نوع الكائن الذى يحتوى علية المتغير قبل تنفيذ بعض الإجراءات علية لأن هناك بعض الكائنات التى لاتدعم استخدام خاصية أو وسيلة معينة. للتغلب على هذة المشكلة، يوفر لنا Visual Basic.NET وسيلتين لتمكيننا من تحديد نوع الكائن المخزن فى متغير: دالة TypeName و عامل TypeOf...Is.

تعيد لنا دالة TypeName عند تنفيذها سلسلة، وتعتبر الخيار المفضل عند الحاجة إلى تخزين أو عرض اسم تصنيف أحد الكائنات، كما يوضحة الكود التالى الذى يستخدم ناتج دالة TypeName معاملا لدالة MsgBox:

```
MsgBox (TypeName(Ctrl))
```

ويعتبر عامل TypeOf...Is هو الخيار الأفضل عند اختبار نوع أحد الكائنات، لأنة

أسرع كثيرا من مقارنة سلسلة باستخدام دالة TypeName. الكود التالى يستخدم هذا العامل داخل عبارة If...Then...Else لعرض رسالة عند مقابلة نوع الكائن المطلوب :

```
If TypeOf Ctrl Is Button Then
   MsgBox ("The control is a button.")
End If
```

ويجب ملاحظة أن عامل TypeOf...Is يعيد القيمة True عندما يكون الكائن من نوع محدد، أو مشتق من نوع محدد. فى Visual Basic .NET، يدخل كل شئ تقريبا تحت تصنيف الكائنات. وعلى هذا الأساس، تعتبر بعض العناصر التى لم تكن تدرج تحت الكائنات فى السابق، كائنات فى تصنيفات Visual Basic.NET. من أمثلة هذة العناصر الأرقام الصحيحة (Integers) و السلاسل (Strings)، وهى عناصر مشتقة من تصنيف System.Object. ولهذا نحصل على القيمة True عند مقارنة نوع Integer مع نوع Object باستخدام عامل TypeOf...Is. المثال التالى يوضح هذة الحقيقة عن طريق عرض رسالتين، تبين إحداهما أن القيمة التى يتم تمريرها إلى دالة CheckType هى من نوع Object، بينما تبين الرسالة الثانية أن نفس هذة القيمة من نوع Integer :

```
Sub CheckType(ByVal InParam)
  If TypeOf InParam Is Object Then
    MsgBox("InParam is an Object")
  End If
  If TypeOf InParam Is Integer Then
    MsgBox("InParam is an Integer")
  End If
End Sub
```

ولتوضيح كيفية استخدام عامل TypeOf...Is و دالة TypeName لتحديد نوع الكائن الذى يتم تمريرة إلى إجراء TestObject، ننفذ التمرين التالى :

١. نكون مشروع نماذج ويندوز ونضيف متحكم Button، متحكم CheckBox، ومتحكم RadioButton إلى النموذج الافتراضي.

٢. نستخدم متحكم Button على النموذج لاستدعاء إجراء TestObject.

٣. نضيف الكود التالى إلى قسم Declarations فى النموذج :

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
```

```
System.EventArgs) Handles Button1.Click
        TestObject()
    End Sub

    Sub ShowType(ByVal Ctrl As Object)
        MsgBox(TypeName(Ctrl))
        If TypeOf Ctrl Is Button Then
            MsgBox("The control is a button.")
        ElseIf TypeOf Ctrl Is CheckBox Then
            MsgBox("The control is a check box.")
        Else
            MsgBox("The object is some other type of control.")
        End If
    End Sub

    Protected Sub TestObject()
        ShowType(Me.Button1)
        ShowType(Me.CheckBox1)
        ShowType(Me.RadioButton1)
    End Sub
```

تحديد خصائص ووسائل التصنيف وقت التشغيل

فى معظم الحالات، يمكننا معرفة خصائص ووسائل أحد الكائنات فى وقت التصميم، وكتابة كود معالجتها على هذا الأساس. من ناحية أخرى، قد لانعرف خصائص ووسائل الكائن وقت التصميم، أو ربما نريد توفير المرونة اللازمة لتمكين المستخدم من تحديد الخصائص أو تنفيذ الوسائل وقت التشغيل.

لتنفيذ وسيلة أو خاصية يتم معرفتها وقت التشغيل، نستخدم دالة CallByName التى تسمح باستخدام سلسلة لتحديد خاصية أو وسيلة. وتأخذ هذة الدالة الصيغة التالية :

```
Result = CallByName (Object, ProcedureName, CallType, Arguments())
```

فى هذة الدالة يمثل معامل Object اسم الكائن الذى نريد العمل معة. ومعامل ProcedureName هو سلسلة تحتوى على إسم الوسيلة أو الخاصية التى سوف يتم استدعاؤها. ويأخذ معامل CallType قيمة ثابتة تمثل نوع الإجراء الذى يتم استدعاؤه. وهناك ثلاثة أنواع من الإجراءات التى يمكن تمريرها إلى هذا المعامل :

```
Microsoft.VisualBasic.CallType.Method
Microsoft.VisualBasic.CallType.Get
Microsoft.VisualBasic.CallType.Set
```

المعامل الأخير Arguments، معامل اختياري يحتوى على مصفوفة من نوع Object تحتوى على معاملات الإجراء السابق تحديدة في معامل ProcedureName.

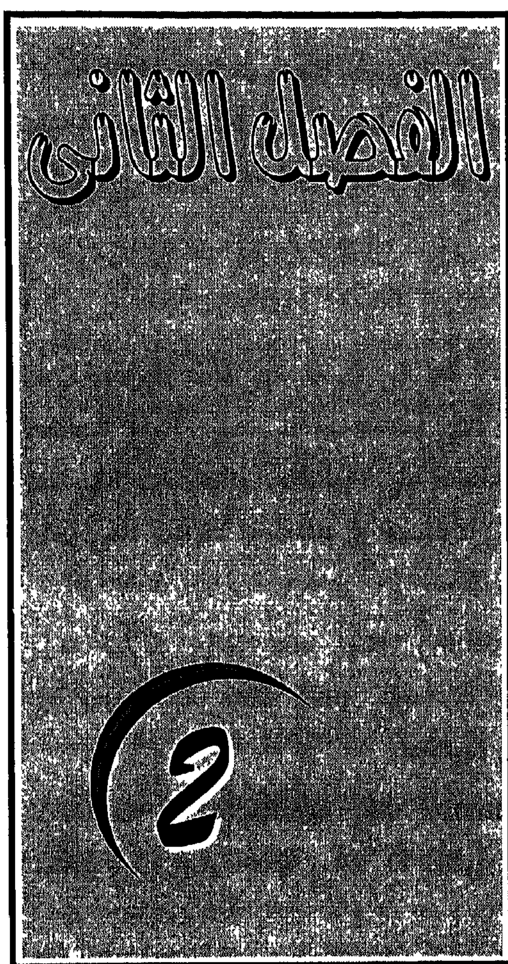
لتوضيح إستخدام هذة الدالة ، نفترض أن هناك تصنيف باسم MathClass يحتوى على دالة جديدة باسم SquareRoot ، كما هو موضح بالكود التالى:

```
Class MathClass
  Function SquareRoot (ByVal X As Double) As Double
    Return Math.Sqrt (X)
  End Function
End Class
```

يمكننا تحديد اسم هذة الوسيلة وتحديد المعامل الذى تتطلبة وقت التشغيل باستخدام دالة CallByName. لتحديد اسم الوسيلة التى نريد تنفيذها، وهى وسيلة SquareRoot، نستخدم مربع النص TextBox2. ولتحديد قيمة المعامل التى نريد تم تمريرة الى هذة الوسيلة، وهو القيمة التى نريد الحصول على جذرها التربيعى، نستخدم مربع النص TextBox1. يمكننا بعد ذلك استخدام الكود التالى لاستدعاء وسيلة SquareRoot لتحديد الجذر التربيعى للقيمة التى يحتوى عليها التعبير الموجود فى مربع النص TextBox1 :

```
Private Sub CallMath()
  Dim Math As New MathClass()
  Me.TextBox1.Text = CStr(CallByName(Math, Me.TextBox2.Text, _
    Microsoft.VisualBasic.CallType.Method, TextBox1.Text))
End Sub
```

فإذا قمنا، على سبيل المثال، بإدخال "64" فى مربع النص TextBox1، "SquareRoot" فى مربع النص TextBox2، ثم إستدعاء إجراء CallMath، فإن الجذر التربيعى الذى سوف نحصل علية فى مربع النص TextBox1 يساوى "8". يتم ذلك عن طريق قيام الكود السابق باستدعاء دالة SquareRoot وتمرير التعبير الذى تحتوى علية خاصية Text فى مربع النص TextBox1 اليها. ويجب أن تكون المعلومات التى يدخلها المستخدم فى مربع النص TextBox2 تمثل اسم صحيح لأحد الوسائل ، كما يجب أن تكون المعاملات التى يتم إدخالها كافية لكى ينجح تنفيذ الكود.



مكونات لغة

Visual Basic.NET

تحتوى الموضوعات التى يشملها هذا الفصل على المكونات الأساسية للغة Visual Basic NET.. هذه المكونات، يحتاج المبرمج إلى فهمها لكى يستطيع كتابة الكود اللازم لتحديد سلوك التطبيق بعد الانتهاء من تكوين الواجهة البينية للتعامل باستخدام النماذج وأدوات التحكم.

عناصر بناء البرنامج

يتكون أي برنامج فى Visual Basic .NET، كما فى غيرها من لغات البرمجة، من أربعة عناصر أساسية: مواقع تخزين البيانات التى يستخدمها البرنامج، أنواع البيانات التى يتم تخزينها، العوامل التى تنفذ عمليات محددة على البيانات، والتعليمات التى تبدأ عمليات التنفيذ. وفيما يلى تفصيل لهذه العناصر الأساسية إلى عناصرها الفرعية التى يمكن استخدام بعضها أو كلها فى بناء البرامج:

- مواقع تخزين البيانات المتغيرة (Variables). تتميز هذه المواقع بقابلية تغيير محتوياتها من البيانات. ويتم تمييز كل موقع باسم محدد لاستخدامه فى البرنامج، كما يتم تمييزه بنوع معين من البيانات.
- مواقع تخزين البيانات الثابتة (Constants). تتميز هذه المواقع بأن البرنامج لا يستطيع تغيير محتوياتها من البيانات. ويتميز كل موقع أيضا، باسم ونوع بيانات معين.
- مواقع تخزين مركبة من مواقع فرعية تستخدم نفس نوع البيانات ولها اسم واحد يميزها. يشمل ذلك المصفوفات (Arrays) والتعدادات (Enumerations).
- أنواع البيانات الأساسية (Elementary Data Types)، التى تشمل البيانات العددية، البيانات الرمزية، بيانات التواريخ، البيانات المنطقية.
- أنواع البيانات المؤلفة (Composite Data Types)، التى تشمل سلاسل الرموز، هياكل البيانات، والتصنيفات.
- العوامل (Operators)، التى تمثل وحدات كود تقوم بإجراء عمليات على عناصر القيم المختلفة. ويمكن تقسيم العوامل إلى عوامل حسابية (Arithmetic Operators)، عوامل المقارنة (Comparison Operators)، عوامل ربط سلاسل الرموز (

(Concatenation Operators)، وعوامل منطقية (Logical Operators).

- التعبيرات (Expressions)، التى تعتبر مجموعات من أسماء البيانات التى تفصل بينها العوامل. وينتج عن تقييم التعبيرات الحصول على قيمة محددة.
- التعليمات (Statements)، التى تأمر Visual Basic بالقيام بعمليات البرنامج المختلفة. وتتكون التعليمات من كلمات مرشدة، أسماء بيانات، وعوامل، وتنقسم إلى مجموعتين أساسيتين: تعليمات الإعلان (Declaration Statements)، والتعليمات المنفذة (Executable Statements).
- الإجراءات (Procedures)، التى تعتبر مجموعات من التعليمات. ويقوم كل إجراء بتنفيذ مهمة محددة، كما يمكن استدعاؤه من أى موقع داخل البرنامج.

الإعلان عن عناصر البرنامج

يرتبط استخدام معظم عناصر البرمجة السابق عرضها، بالإعلان عنها قبل بدء استخدامها. ويشتمل الإعلان عن أحد العناصر تحديد اسم العنصر، نوع البيانات، فترة بقاء العنصر، نطاق رؤية هذا العنصر، وطريقة الوصول إلية.

العناصر التى يتم الإعلان عنها فى برامج Visual Basic .NET

الجدول رقم (٣) يوضح أنواع العناصر التى يجب الإعلان عنها فى برامج Visual Basic ومتطلبات هذا الإعلان:

اسم العنصر	يرتبط بنوع بيانات	يرتبط بفترة بقاء	يرتبط بنطاق رؤية	يرتبط بترخيص وصول
Variable	نعم	نعم	نعم	نعم
Constant	نعم	لا	نعم	نعم
Enumeration	نعم	لا	نعم	نعم
Structure	لا	لا	نعم	نعم
Property	نعم	لا	نعم	نعم
Method	لا	لا	نعم	نعم

اسم العنصر	يرتبط بنوع بيانات	يرتبط بفترة بقاء	يرتبط بنطاق رؤية	يرتبط بترخيص وصول
Procedure	لا	لا	نعم	نعم
Procedure argument	نعم	نعم	نعم	نعم
Function return	نعم	نعم	نعم	نعم
Event	لا	لا	نعم	نعم
Delegate	لا	لا	نعم	نعم
Interface	لا	لا	نعم	نعم
Class	لا	لا	نعم	نعم

جدول ٣

يمثل نوع البيانات (Data Type) فى الجدول السابق، القيم التى يمكن أن يحتفظ بها العنصر، وكيفية تخزين هذه القيم. بينما تمثل فترة البقاء (Lifetime) الفترة التى يكون فيها العنصر متاحا للاستخدام خلال فترة تشغيل البرنامج. ويمثل نطاق الرؤية (Scope) أجزاء البرنامج التى يمكن أن تستخدم هذا العنصر بدون تأهيل اسمه بالكائن الذى بتبعة. وأخيرا، يمثل ترخيص الوصول إمكانية استخدام العنصر بواسطة أجزاء البرنامج الأخرى.

فترة بقاء العنصر

تمثل فترة بقاء العنصر (Life Time) تلك الفترة من الوقت التى يكون فيها العنصر متاحا للاستخدام. والعنصر الذى له فترة بقاء هو المتغير وما يماثلة، مثل معاملات الإجراءات، والقيم العائدة من الدوال. ويحتفظ المتغير خلال فترة بقائه بقيمة معينة يمكن أن تتغير.

العلاقة بين فترة بقاء المتغير ومستوى الإعلان عنة

ويختلف طول فترة بقاء المتغير بناءا على مستوى الإعلان عنة. المتغير الذى يتم الإعلان عنة على مستوى الوحدة، يستمر بقاؤه خلال فترة تشغيل التطبيق بالكامل. والمتغير الذى لا نستخدم كلمة Shared عند الإعلان عنة فى تصنيف أو هيكل بيانات، يتواجد فى صورة نسخة منفصلة مع كل مثل من أمثلة التصنيف أو هيكل البيانات، وتكون فترة بقاء كل

نسخة من المتغير مساوية لفترة بقاء المثل المرتبط به. والمتغير المعلن عنة فى تصنيف أو هيكل بيانات باستخدام كلمة Shared، يكون له فترة بقاء واحدة تستمر طالما استمر تشغيل البرنامج.

بداية فترة بقاء المتغير

تبدأ فترة بقاء المتغير المحلى (Local Variable) عندما يبدأ التطبيق فى تنفيذ الإجراء الذى تم الإعلان عنة فيه. ويتم ضبط المتغير المحلى على القيمة الافتراضية الخاصة به عند بدء تنفيذ الإجراء. المتغيرات العددية، بما فيها المتغيرات من نوع Byte و Char، يتم إعدادها بضبطها على قيمة صفر. بينما تضبط قيمة المتغيرات من نوع Date على منتصف الليل بتاريخ ١/١/١. وتضبط المتغيرات المنطقية على القيمة False؛ وتضبط متغيرات المراجع، مثل السلاسل ، المصفوفات، والكائنات على القيمة Nothing. ويجرى إعداد كل عضو من أعضاء هياكل البيانات كما لو أنة متغير مستقل. وبالمثل يتم إعداد كل عنصر من عناصر المصفوفات بطريقة منفصلة. وفى حالة تخصيص قيمة ابتدائية للمتغير، يجرى تخصيصها له عند تنفيذ تعليمات الإعلان عنة. والمتغيرات التى يتم الإعلان عنها داخل مجمع كود بداخل أحد الإجراءات، يجرى ضبطها على قيمتها الافتراضية عند الدخول إلى الإجراء. ولا يعتمد ذلك على تنفيذ أو عدم تنفيذه مجمع الكود الذى تتواجد به.

نهاية فترة بقاء المتغير

عند الانتهاء من تنفيذ أحد الإجراءات، لا يتم الاحتفاظ بقيم المتغيرات التى يحتوى عليها، ويقوم النظام باسترجاع الذاكرة التى كانت مخصصة لهذه المتغيرات. وعند بدء تنفيذ الإجراء مرة أخرى، يعاد تكوين المتغيرات وإعدادها من جديد. وعند انتهاء مثل(Instance) من تصنيف أو هيكل بيانات، تفقد المتغيرات غير المشتركة القيم التى تحتفظ بها. ويقوم كل مثل جديد من التصنيف أو من هيكل البيانات بتكوين كل المتغيرات غير المشتركة (nonshared elements) وإعدادها من جديد. بينما يتم الاحتفاظ بالعناصر المشتركة إلى أن يتوقف تنفيذ البرنامج.

تحديد فترة بقاء المتغيرات

يمكن جعل فترة بقاء المتغير المحلى أطول من فترة تنفيذ الإجراء الذى تم إعلانه به عن طريق استخدام كلمة Static. فإذا كان الإجراء يقع داخل وحدة ، فإن المتغير الساكن تستمر فترة بقائه طالما استمر تنفيذ البرنامج. وعند الإعلان عن المتغير الساكن داخل إجراء

فى تصنيف، فإن فترة بقائه تعتمد على ما إذا كان الإجراء مشترك أو غير مشترك. إذا تم الإعلان عن الإجراء باستخدام Shared، تستمر فترة بقاء المتغير الساكن إلى انتهاء التطبيق. وإذا تم الإعلان عن الإجراء بدون استخدام Shared، تصبح المتغيرات الساكنة أعضاء فى أمثلة التصنيف، وتصبح فترة بقائها مثل فترة بقاء مثل التصنيف. فى المثال التالى، تقوم دالة Totals بحساب الإجمالي المتحرك عن طريق إضافة قيمة جديدة إلى القيمة التى تم تخزينها فى المتغير الساكن SalesTot، الذى يحتفظ بالقيم المتراكمة السابقة:

```
Function Totals (ByVal Num As Integer) As Integer
        Static SalesTot As Integer
        SalesTot = SalesTot + Num
        Totals = SalesTot
End Function
```

نطاق رؤية عناصر البرامج

نطاق الرؤية (Scope) الخاص بالعناصر المعلن عنها يتكون من أجزاء الكود التى يمكنها استخدام هذه العناصر بدون تأهيل أسمائها أو جعلها متاحة من خلال استخدام عبارة Imports. ويمكن رؤية العنصر على مستوى من المستويات التالية:

- نطاق رؤية مجمع الكود (Block Scope).
- نطاق رؤية الإجراء (Procedure Scope).
- نطاق رؤية وحدة الكود (Module Scope).
- نطاق رؤية نطاق الأسماء (Namespace Scope).

نطاق رؤية مجمع الكود

يمكن تعريف مجمع الكود بأنة مجموعة من التعليمات التى تنتهي بعبارة End، Else، Loop، أو Next. على سبيل المثال، الكود الموجود داخل For...Next أو الكود الموجود داخل If...Then...Else...End If. والعناصر المعلنة داخل مجمع كود يمكن استخدامها فقط داخل ذلك المجمع. فى المثال التالى، نطاق رؤية المتغير Cube هو مجمع الكود الذى يقع بين عبارات If و End If، و لايمكن استخدام المتغير Cube خارج هذا المجمع :

```
If N < 1291 Then
  Dim Cube As Integer
  Cube = N ^ 3
End If
```

ومع أن نطاق رؤية العنصر يقع داخل إطار مجمع الكود المذكور، إلا أن فترة بقاء العنصر تستمر أثناء فترة بقاء الإجراء المعلن به. ولذلك، يستمر هذا المتغير فى الاحتفاظ بالقيمة المخزنة به طالما استمر تنفيذ كود الإجراء.

نطاق رؤية الإجراء

لا يمكن رؤية العنصر المعلن عنة داخل إجراء خارج ذلك الإجراء. وتعرف العناصر المعلنة على هذا المستوى بالعناصر المحلية. ويمكن الإعلان عنها باستخدام عبارات Dim، و Static. وهناك علاقة قوية بين نطاق رؤية مجمع الكود وبين نطاق رؤية الإجراء. عند الإعلان عن عنصر داخل إجراء ولكن خارج مجمع كود داخل هذا الإجراء، يدخل هذا العنصر فى نطاق رؤية مجمع الكود أيضا.

نطاق رؤية وحدة الكود

ينطبق تعبير مستوى الوحدة بالتساوي على وحدات الكود (Modules)، التصنيفات (Classes)، وهياكل البيانات (Structures). ويمكن الإعلان عن العناصر على هذا المستوى عن طريق وضع عبارات الإعلان خارج أى إجراء أو مجمع كود داخل الوحدة، التصنيف، أو هيكل البيانات. وعند الإعلان عن أحد العناصر على مستوى الوحدة، فإن ترخيص الوصول (Accessibility) الذى يتم اختياره هو الذى يحدد نطاق الرؤية. كما أن منطقة الأسماء التى تحتوى على الوحدة، التصنيف، وهيكل البيانات تؤثر أيضا على نطاق رؤية العناصر المعلن عنها.

العناصر التى يكون ترخيص الوصول إليها خاص (Private) تكون متاحة للاستخدام بواسطة كل الإجراءات فى الوحدة، ولا تكون متاحة أمام الوحدات الأخرى. وتساوى كلمة Dim على مستوى الوحدة كلمة Private عند عدم استخدام أى كلمات وصول أخرى. فى المثال التالى، متغير strMsg يكون متاحا لكل الإجراءات فى الوحدة، ولهذا يقوم الإجراء الأول فى المثال، بوضع قيمة به. وعند استدعاء الإجراء الثانى، يقوم بعرض محتويات نفس المتغير فى مربع رسالة:

```
Private strMsg As String
Sub InitializePrivateVariable ()
   strMsg = "This variable cannot be used outside this module."
End Sub
Sub UsePrivateVariable()
   MsgBox (strMsg)
```

```
End Sub
```

نطاق رؤية منطقة الأسماء

عند الإعلان عن عنصر على مستوى الوحدة باستخدام كلمات Friend أو Public، يصبح هذا العنصر متاحا لكل الإجراءات الموجودة بنفس نطاق البيانات التى يتبعها العنصر المعلن عنة. وعند عدم استخدام عبارات Namespace، يصبح كل شئ يخص المشروع فى منطقة أسماء واحدة. فى هذه الحالة يمكن النظر إلى نطاق رؤية منطقة الأسماء على أنة نطاق رؤية المشروع بالكامل. وعند استخدام كلمة Public فى وحدة (Module)، تصنيف (Class)، أو هيكل بيانات (Structure)، يصبح العنصر متاحا أمام أى مشروع يرجع إلى المشروع الذى تم إعلانه به.

تراخيص الوصول

ترخيص الوصول (Accessibility) للعنصر المعلن عنة تحدد القدرة على استخدامه أو إمكانية القراءة والكتابة بهذا العنصر. وتتقرر هذه القدرة على أساس ترخيص الوصول الخاص بحاوية العناصر، بالإضافة إلى طريقة الإعلان عن العنصر. فإذا كانت الحاوية التى يوجد بها العنصر لايمكن الوصول إليها، فإن جميع العناصر التى تحتويها لايمكن الوصول إليها من خارج الحاوية أيضا، حتى فى حالة استخدام ترخيص Public فى الإعلان عن هذه العناصر. على سبيل المثال، المتغيرات العامة داخل هيكل بيانات خاص، يمكن الوصول إليها من داخل الهيكل فقط.

ترخيص Public

عند استخدام كلمة Public فى عبارة Dim، فإن العناصر المعلن عنها يمكن الوصول إليها من أى مكان داخل نفس المشروع، من المشروعات الأخرى التى تحتوى على مراجع لذلك المشروع، ومن وحدة التجميع (Assembly) المبنية من ذلك المشروع. يوضح الكود التالى كيفية الإعلان باستخدام كلمة Public:

```
Public Class ClassForEverybody
```

ويمكن استخدام Public على مستوى الوحدة، منطقة الأسماء، أو الملف. يعنى ذلك أننا نستطيع الإعلان عن عنصر عام (Public Element) داخل ملف كود أو داخل وحدة، تصنيف، أو هيكل بيانات، ولكن ليس داخل إجراء.

ترخيص protected

يمكن استخدام كلمة Protected فقط على مستوى التصنيف، وفقط عند الإعلان عن عضو بتصنيف. وعند استخدام كلمة Protected فى الإعلان عن العناصر، فإن هذه العناصر يمكن الوصول إليها فقط من داخل التصنيف الذى تم الإعلان به أو من داخل تصنيف مشتق من ذلك التصنيف. ويأخذ الإعلان باستخدام Protected، الصيغة التالية:

```
Protected Class ClassForMyHeirs
```

ترخيص Friend

نستخدم كلمة Friend للإعلان عن العناصر التى يمكن الوصول إليها من داخل نفس المشروع، ولكن ليس من خارج المشروع. الكود التالى، يوضح كيفية الإعلان عن عنصر باستخدام كلمة Friend:

```
Friend StringForThisProject As String
```

ويمكن استخدام كلمة Friend فقط على مستوى الوحدة، مستوى منطقة الأسماء، أو مستوى الملف. يعنى ذلك أننا نستطيع الإعلان عن عنصر صديق فى ملف كود أو داخل وحدة، تصنيف، أو هيكل بيانات، و ليس داخل إجراء.

ترخيص Protected Friend

يمكن استخدام Protected Friend على مستوى التصنيف فقط، وعند الإعلان فقط عن عضو فى تصنيف. وعن طريق استخدام هاتين الكلمتين يمكن الوصول إلى العناصر المعلن عنها من داخل المشروع أو من داخل التصنيفات المشتقة من التصنيف الذى تم به الإعلان أو من كليهما. يوضح الكود التالى كيفية الإعلان عن عنصر باستخدام هاتين الكلمتين:

```
Protected Friend StringForProjectAndHeirs As String
```

ترخيص Private

تستخدم كلمة Private للإعلان عن العناصر التى يمكن الوصول إليها من داخل نفس الوحدة، التصنيف، أو هيكل البيانات. يعرض الكود التالى كيفية الإعلان عن عنصر باستخدام Private:

```
Private NumberForMeOnly As Integer
```

ويمكن استخدام Private فقط على مستوى الوحدة، منطقة الأسماء، أو مستوى الملف. يعنى ذلك أننا نستطيع الإعلان باستخدام كلمة Private فى ملف كود، أو داخل وحدة، تصنيف، أو هيكل بيانات، وليس داخل إجراء.

المتغيرات (Variables)

نحتاج فى الغالب إلى تخزين القيم بصفة مؤقتة عند تنفيذ العمليات الحسابية. على سبيل المثال، قد نحتاج إلى حساب مجموعة من القيم، المقارنة بينها، وتنفيذ بعض العمليات على هذه القيم، على أساس نتيجة المقارنة بينها. فى مثل هذه الحالة، نحتاج إلى تخزين القيم بعد احتسابها لكى يمكن إجراء باقى العمليات عليها.

ومثل غيرها من لغات الكمبيوتر الأخرى، تستخدم Visual Basic المتغيرات(Variables) لتخزين القيم المؤقتة. والمتغير له اسم يستخدم للإشارة إلى القيمة التى يحتوى عليها، كما يرتبط بنوع بيانات يمكن تخزينها به. ونستخدم عبارات التخصيص لتحديد قيمة لمتغير أو القيام بإجراء العمليات الحسابية وتخصيص القيمة الناتجة لمتغير. ويتحكم استخدام عبارات Options فى ضرورة الإعلان عن المتغيرات قبل استخدامها أو استخدامها بدون الإعلان عنها.

عبارات Options

يرتبط استخدام المتغيرات والثوابت ارتباطا وثيقا بعبارات Option، التى تضبط القواعد التى يبنى عليها الكود الذى يأتى بعدها فى البرنامج. وتساعد هذه العبارات على منع الأخطاء اللغوية والمنطقية التى قد تحدث بالبرنامج، وتتكون من عبارة Option Explicit، عبارة Strict Option، وعبارة Option Compare.

عبارة Option Explicit

يتطلب Visual Basic .NET فى الأصل، الإعلان الصريح عن المتغيرات التى يستخدمها البرنامج، ولكنة يسمح لنا بتغيير هذا الوضع والإعلان ضمنيا عن المتغير. يمكن الإعلان عن متغير ضمنيا بطريقة مماثلة لما يلى :

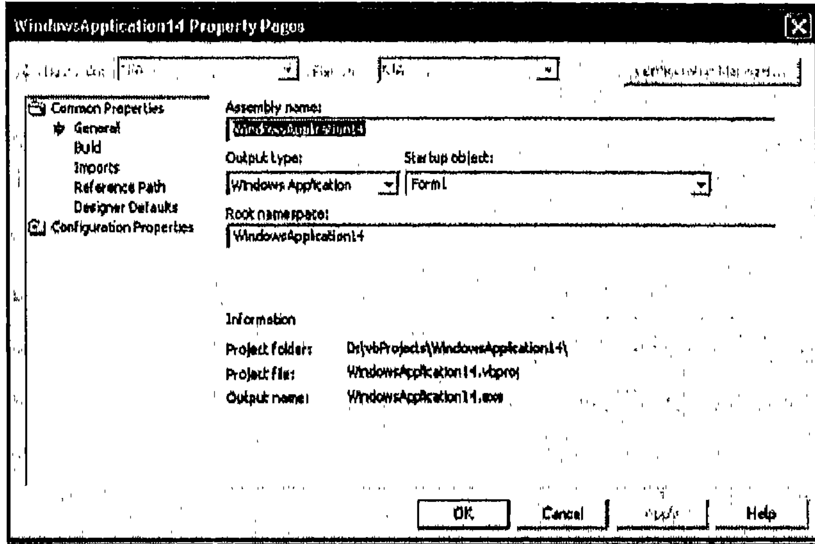
```
MyVariable = 10
```

يترتب على العبارة السابقة ، قيام برنامج الترجمة (Compiler) بافتراض أن MyVariable متغير ، طالما لم يتم الإعلان عنة فى صورة كائن أو خاصية، وأن القيمة المخصصة له تساوى 10. كما يفترض برنامج الترجمة أن نوع البيانات فى العبارة السابقة هو Object، طالما لم يتم تحديد نوع البيانات. وعلى العكس من ذلك ، يتطلب الإعلان الصريح إعلان ذلك المتغير قبل تخصيص قيمة له. يوضح الكود التالى مثالا للإعلان الصريح عن متغير:

```
Dim intVar As Integer
```

يترتب على إدراج عبارة Option Explicit، ظهور رسالة خطأ أثناء بناء البرنامج عند محاولة استخدام متغير لم يتم الإعلان عنة. ويمكن تنفيذ هذا الاختيار باستخدام مربع حوار Property Pages، الموضح بالشكل رقم (٦). وللحصول على مربع الحوار المذكور، ننقر بزر الماوس الأيمن على المشروع فى مربع Solution Explorer ثم نختار Build من مربع Common Properties فى الجانب الأيسر. بالنظر إلى المربع السابق ، نرى أن Option Explicit قد تم ضبطها افتراضيا على On. ويمكن تغيير هذا الاختيار فى المربع المذكور ثم النقر على زر Ok. ويمكَن أيضا ضبط هذا الخيار باستخدام الكود ، بجعل عبارة Option Explicit أول سطر فى وحدة الكود قبل الإعلان عن التصنيف :

```
Option Explicit On
Public Class Form1
    Inherits System.Windows.Forms.Form
```



شكل رقم ٦

عبارة Option Strict

يوجد بالشكل رقم (٦) خيارا أخرا، هو Option Strict. يماثل خيار Option Strict خيار Option Explicit فى ضرورة الإعلان عن المتغير. ولكن على خلاف خيار Option Explicit، يتطلب هذا الخيار الإعلان عن نوع بيانات المتغير أو الثابت. ويمكن تفعيل هذا الخيار أو عدم تفعيلة، مثل الخيار السابق. وعند استخدام هذا الخيار فى الكود الذى نقوم

بكتابته، لا نحتاج إلى استخدام خيار Option Explicit. ونستخدم Option Strict بنفس الطريقة التى نستخدم بها Option Explicit، كما يتضح من الكود التالى :

```
Option Strict On
Public Class Form1
    Inherits System.Windows.Forms.Form
```

وفى الأصل، يكون Option Strict غير فعال فى البرامج، ويجب تفعيلة عند الرغبة فى استخدامه. وحتى فى حالة عدم تفعيلة ، نحتاج إلى الإعلان عن نوع القيمة. وإذا لم نعلن نوع القيمة، يقوم برنامج الترجمة باختيار نوع لها بالنيابة عنا. ويعتمد نوع البيانات الذى يختاره برنامج الترجمة على القيمة التى نقوم بإدخالها. على سبيل المثال، عند إدخال القيمة التالية، سوف يكون نوع البيانات المختار Integer:

```
Dim x = 100
```

وسوف يكون نوع البيانات المختار Double، عند إدخال القيمة التالية:

```
Dim x = 100.1
```

عبارة Option Compare

تستخدم Option Compare على مستوى الملف للإعلان عن طريقة المقارنة الافتراضية المستخدمة للمقارنة بين سلاسل البيانات. وتأخذ هذه العبارة الصيغة التالية :

```
Option Compare {Binary | Text}
```

فى هذه العبارة، يمكن استخدام Binary لجعل المقارنة بين السلاسل تقوم على أساس التمثيل الثنائى للبيانات. كما يمكن استخدام Text للمقارنة بين السلاسل على أساس النصوص. وكما هو الحال فى عبارات الاختيار السابقة، يجب أن تكون Option Compare أول عبارة فى الملف. وتعتبر قيم Binary و Text اختيارية. وإذا لم يتم اختيار إحداها فى عبارة Option Compare، فإن القيمة الافتراضية هى String.

يترتب على اختيار Option Compare Binary، تحقق التعبير التالى:

```
"AAA" is less than "aaa"
```

ويترتب على اختيار Option Compare Text، تحقق التعبير التالى:

```
"AAA" is equal to "aaa"
```

الإعلان عن المتغير

نقوم بالإعلان عن المتغير (Declaring Variable) بهدف تكوينه وتحديد الاسم والمواصفات الخاصة به. ويتحكم مستوى الإعلان عن المتغير وكلمات الوصول المستخدمة فى

تحديد فترة بقاء المتغير، نطاق رؤيته، وتراخيص الوصول إلية. ويأخذ التركيب اللغوى للإعلان عن المتغير الصورة التالية :

[Access Specifier] [Variable Name] As [Data Type]

يتكون هذا التركيب اللغوى من عدد من الأجزاء: ترخيص الوصول(Access Specifiers)، اسم المتغير (Variable Name)، ونوع البيانات (Data Type).

نوع البيانات

كما سبق إيضاحية ، لا نحتاج إلى تحديد نوع البيانات (Data Type) عندما لا نستخدم Option Strict، لأن برنامج الترجمة يقوم بتحديد النوع بناءا على القيمة التى يتم إدخالها. ويمكننا اختيار نوع مناسب من البيانات عند الإعلان عن متغير. ويستخدم Visual Basic .NET الكثير من أنواع البيانات التى سوف يجرى مناقشتها بعد ذلك.

تحديد اسم للمتغير

هناك بعض القواعد التى تتعلق بتحديد اسم المتغير (Variable Naming)، نلخصها فيما يلى:

- يجب أن يبدأ الاسم بحرف من حروف الهجاء أو شرطة أسفل السطر.
- لا يجب أن يحتوى الاسم على فراغات داخلية.
- لايجب أن يتعدى طول الاسم ٢٥٥ حرف.
- لا يمكن أن يكون الاسم أحد الكلمات المفتاحية (Keywords).
- لا يمكن تكرار اسم المتغير داخل نفس نطاق الرؤية الخاص به.

ويجب استخدام إحدى طرق التسمية (Naming Conventions) الشائعة الاستخدام. ويقصد بمصطلح Naming Convention، الطريقة الثابتة ذات الدلالة فى التسمية. إحدى هذه الطرق تستخدم ثلاثة حروف صغيرة للدلالة على نوع المتغير يأتى بعدها اسم يشير إلى الغرض من الكائن الذى نقوم بتسميته. يعرض الجدول رقم (٤) مقدمات أسماء، كما يعرض الأنواع المرتبطة بها :

نوع البيانات	البادئة
Boolean	bin
Byte	byt

نوع البيانات	البادئة
Char	chr
Date	dat
Decimal	dec
Double	dbl
Integer	int
Long	lng
Object	obj
Short	sht
Single	sng
String	str

جدول ٤

تراخيص الوصول

تعتمد تراخيص الوصول (Access Specifiers) وتأثيرها على موقع الإعلان عن المتغيرات. حيث يمكن الإعلان عن المتغير على مستوى وحدة الكود أو داخل أحد الإجراءات. عند الإعلان عن متغير محلى (Local Variable)، نستخدم كلمات Dim أو Static. حيث يتشابه نطاق رؤية المتغير عند الإعلان عنة باستخدام Dim أو Static داخل إجراء. ولكن هناك اختلاف بين Dim وبين Static يتعلق بفترة بقاء المتغير فى البرنامج. لتوضيح الفرق بين Dim وبين Static، نبدأ تطبيق نماذج ويندوز جديد ونضع الكود التالى داخل إجراء معالجة حدث النقر على النموذج الأساسى:

```
    Private Sub Form1_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles MyBase.Click
        Dim i As Integer
        MessageBox.Show(Str (i))
        i = i + 1
    End Sub
```

يترتب على تشغيل التطبيق السابق الحصول دائما على القيمة صفر. وعند تغيير Dim إلى Static، سوف نحصل على قيمة متزايدة فى كل مرة نقوم فيها بالنقر على النموذج. يترتب على استخدام Static عند الإعلان عن متغير، تذكر المتغير أخر قيمة.

وعند الإعلان عن متغير على مستوى وحدة الكود، يمكن استخدام Dim، Private،

Public، أو Shared. وليس هناك فرق بين ترخيص الوصول Dim وترخيص الوصول Private . والسبب فى استخدام كلمة Dim هو تحقيق التوافق مع الإصدارات السابقة من Visual Basic. ولا يمكن استخدام Private للإعلان عن متغير داخل أحد الإجراءات. ويترتب على استخدام Private على مستوى وحدة الكود، أنة يصبح متاحا بالوحدة التى تم إعلانه بها فقط، وغير منظور بالنسبة لوحدات الكود الأخرى فى التطبيق. يعنى ذلك، أن وحدات الكود الأخرى فى التطبيق لا يمكنها استخدام هذا المتغير. غير أنة يمكننا الوصول إلى هذا المتغير من خلال وحدة كود أخرى، مثل إجراء Property Get.

وعند استخدام كلمة Public فى تعريف متغير، يصبح هذا المتغير متاحا للاستخدام فى جميع وحدات المشروع باستخدام تركيب لغوى مماثل للتركيب التالى :

[FormObjectVariable].[Variable Name]

وتستخدم كلمة Shared لإتاحة استخدام المتغير على مستوى التصنيف بدلا من مستوى كائن من كائنات التصنيف. وفى الأصل، تعتبر المتغيرات التى تستخدم كلمة Shared، خاصة. ولكن يمكننا تحويلها إلى عامة باستخدام كلمة Public، كما يتضح من الكود التالى :

Public Shared [variable name] As [Data Type]

وعندما لا يستخدم المتغير كلمة Shared، يستخدم كل كائن من كائنات التصنيف نسخة خاصة به من المتغير. وعلى العكس من ذلك، عند الإعلان عن متغير باستخدام Shared، فأن جميع كائنات التصنيف تستخدم ذات المتغير. ولتوضيح استخدام المتغيرات التى نستخدم كلمة Shared فى الإعلان عنها، نقوم بتنفيذ التمرين التالى:

١. نقوم بتكوين مشروع جديد.

٢. نعلن عن المتغيرات التالية ونخصص لها القيمة صفر فى قسم الإعلان بالنموذج الإفتراضى:

```
Private intPrivate As Integer = 0
Public intPublic As Integer = 0
Shared intShared As Integer = 0
```

٣. نضع متحكم Button على النموذج الإفتراضى ونضبط خاصية Name على btnShowForm2 ونضبط خاصية Text على Show Form2.

٤. نضع عدد ٦ من أدوات تحكم Label على النموذج ، كما هو محدد فى الجدول رقم (٥).

خاصية Text	خاصية Name
Private	lblPrivateText
	lblPrivateValue
Public	lblPublicText
	lblPublicValue
Shared	lblSharedText
	lblSharedValue

جدول ٥

٥. نضع الكود التالى فى إجراء معالجة حدث btnShowForm2_Click :

```
    Private Sub btnShowForm2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnShowForm2.Click
            Dim frm As New Form1()
            intPrivate = intPrivate + 1
            intPublic = intPublic + 1
            intShared = intShared + 1
            lblPrivateValue.Text = frm.intPrivate
            lblPublicValue.Text = frm.intPublic
            lblSharedValue.Text = frm.intShared
        End Sub
```

٦. نقوم بتشغيل التطبيق بالضغط على مفتاح F5.

٧. فى كل مرة ننقر على زر Show Form، تتزايد قيمة Shared variable بمقدار واحد فى كل مرة، بينما تبقى قيمة Public Variable وقيمة Private Variable بدون تغيير.

متغيرات الكائنات

يمكن استخدام المتغيرات لتخزين المراجع إلى الكائنات، بالإضافة إلى استخدامها فى تخزين القيم. يطلق على هذا النوع من المتغيرات Object Variables. ونقوم بتخصيص كائن لمتغير لنفس الأسباب التى تجعلنا نخصص قيمة لمتغير، مثل سهولة تذكر واستخدام أسماء المتغيرات، إمكانية تغيير محتويات المتغير لكى يشير إلى كائن أخر. وبمجرد تخصيص كائن لمتغير، يمكننا معاملة هذا المتغير بنفس الطريقة التى نعامل بها الكائن الذى يشير إلية. على سبيل المثال، يمكن ضبط أو استرجاع الخصائص الخاصة بالكائن أو استخدام أى من وسائلة، كما يتضح من الكود التالى:

```
Dim ctrlBal As Control
ctrlBal = frmAccountDisplay.ActiveForm.ActiveControl
ctrlBal.Text = "Test"
ctrlBal.SetLocation (100, 100)
ctrlBal.Show ()
```

الإعلان عن متغير كائن

نستخدم تعليمات الإعلان المعتادة للإعلان عن متغير كائن، ونوع البيانات فى هذه الحالة، يكون إما كلمة Object أو تصنيف أحد الكائنات (Object Class). ويمكن تعريف تصنيف الكائن بأنة التصنيف الذى يتم استنساخ الكائن منة. وعند الإعلان عن متغير من نوع تصنيف كائن معين، يمكن لهذا المتغير الوصول إلى كل الوسائل والخصائص التى يكشف عنها ذلك التصنيف. وعند الإعلان عن متغير من نوع Object، يمكن لهذا المتغير استخدام أعضاء تصنيف Object فقط، إلا إذا تم ضبط Option Strict على القيمة Off. ويأخذ الإعلان عن متغير الكائن الصيغة التالية:

Dim *variablename* As [New] {*objectclass* | Object}

ويمكن أيضا استخدام كلمات Protected ، Friend، Protected Friend، Private، Shared، أو Static فى عبارة الإعلان، كما يتضح من نماذج الكود التالية:

```
Private ObjA As Object
Static ObjB As Label
Dim ObjC As System.Buffer
```

وعند تفعيل عبارة Option Strict، يمكن لمتغير كائن الوصول فقط إلى وسائل وخصائص التصنيف المستخدم فى الإعلان عنة. المثال التالى، يوضح ذلك:

```
Imports System.Windows.Forms
Dim P As Object
Dim Q As Label
P = New Label
Q = New Label
Dim J, K As Integer
J = P.Left
K = Q.Left
```

فى هذا المثال، يمكن للمتغير P استخدام أعضاء التصنيف الخاص بالكائن، وهذه الأعضاء لا تشتمل على خاصية Left، ولذلك نحصل على خطأ عند محاولة الحصول على قيمة هذه الخاصية باستخدام متغير P. من الناحية الأخرى، المتغير Q تم الإعلان عنة ليكون من نوع Label، ولهذا يمكنة استخدام كل الوسائل والخصائص فى تصنيف Label،

الذى يوجد فى منطقة أسماء System.Windows.Forms.

تخصيص قيمة لمتغير الكائن

نستخدم عبارة تخصيص لتخزين قيمة فى متغير كائن. وفى هذه الحالة، يمكن تخصيص مرجع كائن أو كلمة Nothing. الكود التالى يستخدم عبارة التخصيص الأولى لوضع مرجع كائن فى المتغير، بينما يستخدم العبارة الثانية فى الفصل بين المتغير ومرجع الكائن:

```
MyObject = YourObject
MyObject = Nothing
```

وعند بدء تشغيل الكود، يتم إعداد متغيرات الكائنات بوضع القيمة Nothing بها، إلا إذا كانت الإعلانات تحتوى على تخصيص مراجع كائنات للمتغيرات. ويمكن الجمع بين الإعلان عن متغير الكائن وبين التخصيص عن طريق استخدام كلمة New. العبارات التالية تقوم بالإعلان عن متغيرات كائنات وتخصيص كائنات لها.

```
Dim NextBuffer As New System.Buffer
Dim Ver As New System.Version(Major, Minor, Revision)
```

ويمكن استخدام القيمة Nothing لمعرفة ما إذا كان متغير الكود يحتوى على مرجع كائن أم لا، كما يتضح من الكود التالى بيانه:

```
If Not MyObject Is Nothing Then
End If
```

وتستخدم كلمة Me لتكون بمثابة متغير كائن يشير إلى المثل الجاري استخدامه من أحد التصنيفات. وإذا كان أحد الإجراءات غير مشترك (non-shared)، يمكنة استخدام Me للحصول على مؤشر (Pointer) إلى المثل الجاري تنفيذه من التصنيف.

الثوابت

الثوابت (Constants) هى طريقة لاستخدام أسماء ذات دلالة بدلا من قيم لا تتغير. وتقوم الثوابت بتخزين هذه البيانات التى لا تتغير خلال فترة تشغيل البرنامج. ويؤدى استخدام الأسماء بدلا من الأرقام والسلاسل إلى زيادة وضوح الكود وسهولة فهمة. ويتم الإعلان عن الثابت باستخدام عبارة Const مع نفس الإرشادات الموضحة عند الإعلان عن المتغيرات. وعند تفعيل Option Strict، يجب التحديد الصريح لنوع بيانات الثابت.

ويماثل نطاق رؤية الثابت نطاق رؤية المتغير الذى يتم الإعلان عنة فى نفس المكان.

ويمكن حصر نطاق رؤية الثابت داخل إجراء بالإعلان عنة داخل ذلك الإجراء. ولجعل أحد الثوابت متاحا خلال كامل التطبيق، نعلن عنة باستخدام كلمة Public فى قسم الإعلانات بالتصنيف. ويمكن أن تكون الثوابت داخلية بالنسبة لأدوات التحكم والمكونات التى نستخدمها، كما يمكن أن يقوم المستخدم بتكوينها.

الإعلان عن الثوابت

يماثل الثابت (Constants) المتغير (Variables)، فيما عدا أن الثابت لا يمكن أن يتغير أثناء تشغيل البرنامج. ويماثل التركيب اللغوى المستخدم للإعلان عن الثابت التركيب المستخدم للإعلان عن المتغير، كما يتضح من صيغة الكود التالية :

```
[Access Specifier] Const [Constant Name] As [Data Type] = [literal]
```

والفرق الأساسى بين الإعلان عن المتغير والإعلان عن الثابت هو أن الإعلان عن الثابت يجب أن يشتمل على تحديد قيمة هذا الثابت. ويرجع السبب فى ذلك إلى أن الثابت لا يمكن تغييره بعد الإعلان عنة. بالإضافة إلى ذلك، يجب أن تكون قيمة الثابت عدد أو سلسلة حروف مجردة (Literal). يعنى ذلك عدم إمكانية استخدام متغيرات لتحديد قيمة الثابت، ولكن يمكن استخدام تعبير على يمين عامل (=) ينتج عنة عدد أو سلسلة حروف. و لايجب أن يحتوى ذلك التعنبير على استدعاء لدالة، ولكن يمكن أن يحتوى على ثوابت سبق إعلانها. وفيما عدا ضبط قيمة الثابت عند الإعلان عنة، تنطبق باقى القواعد التى تحكم المتغير على الثابت.

ويمكن الإعلان عن الثابت داخل إجراء ، كما يمكن الإعلان عنة على مستوى وحدة الكود. وينسحب تأثير استخدام Public، Private، و Option Strict على الثوابت مثل تأثيرها على المتغيرات. غير أن أنواع البيانات المستخدمة مع الثوابت أقل من تلك المستخدمة مع المتغيرات. يمكن أن يكون الثابت من نوع Boolean، Byte، Char، Date، Decimal، Double، Integer، Long، Short، Single، أو String. ولا يمكن أن تكون الثوابت من نوع Object. وتختلف طريقة التسمية المستخدمة مع الثوابت عن تلك المستخدمة مع المتغيرات. مع الثوابت، نستخدم أسماء وصفية ذات حروف كبيرة تفصل بينها شرطة أسفل السطر ، ولا نستخدم الحروف البادئة التى تدل على نوع البيانات، كما هو الحال مع المتغيرات.

أنواع بيانات الثوابت

عندما يكون Option Strict فعالا، يجب الإعلان الصريح عن أنواع بيانات (Data Types) الخاصة بالثوابت. وعندما لا يكون Option Strict فعالا، يستخدم برنامج ترجمة الكود (the Compiler) نوع بيانات التعبير المستخدم فى إعداد الثابت. حيث يتم تحويل القيمة العددية الصحيحة إلى نوع البيانات Integer، تحويل أعداد العلامة العشرية المتحركة إلى نوع البيانات Double، وتحويل كلمات True و False إلى Boolean. ويمكن فرض نوع بيانات معين على أحد القيم المجردة بأن نلحق به حرف يدل على النوع، أو عن طريق وضعة بين حروف محيطة. لتوضيح استخدام حروف النوع والحروف المحيطة لفرض أنواع بيانات على قيم الثوابت، نعرض الأمثلة التالية، :

```
Option Strict Off
```

الكود التالى يفرض على قيمة الثابت أن تكون من نوع Char.

```
Public Const MyCharacter = "a"C
```

الكود التالى يفرض على قيمة الثابت أن تكون من نوع DateTime.

```
Public Const MyDate = #01/15/01#
```

ويبين الجدول رقم (٦) حروف النوع والحروف المحيطة المتاحة فى Visual Basic:

نوع البيانات	الحروف المحيطة	حروف الأنواع
Boolean	(None)	(None)
Byte	(None)	(None)
Char	"	C
Date	#	(None)
Decimal	(None)	D or @
Double	(None)	R or #
Integer	(None)	I or %
Long	(None)	L or &
Short	(None)	S
Single	(None)	F or !
String	"	(None)

جدول ٦

التعدادات

يوفر التعداد (Enumeration) طريقة مناسبة للعمل مع فئات من الثوابت ذات العلاقات وربط القيم الثابتة بأسماء. على سبيل المثال، يمكن الإعلان عن تعداد من الأرقام الصحيحة المرتبطة مع أيام الأسبوع، ثم استخدام أسماء الأيام بدلا من الأرقام الصحيحة.

ويتم الإعلان عن التعداد فى قسم الإعلان بالوحدة أو التصنيف، باستخدام عبارة Enum. ويعتبر التعداد نوع بيانات له اسم، ونوع بيانات أساسي، ومجموعة من الحقول، كل منها يمثل أحد الثوابت. ويجب أن يكون الاسم من الأسماء الصالحة للاستخدام فى Visual Basic .NET. كما يجب أن يكون نوع البيانات الأساسى من أنواع الأرقام الصحيحة، مثل Byte، Short، Long، أو Integer. ويعتبر Integer هو النوع الإفتراضى. ويمكن تخصيص نفس القيمة لعدد من الحقول. ويمكن استخدام عبارة تخصيص للتحديد الصريح لقيم الثوابت فى أحد التعدادات. ويمكن استخدام كل القيم الصحيحة، بما فيها القيمة السالبة، فى التعدادات.

الإعلان عن التعدادات

نستخدم عبارة Enum فى قسم الإعلانات بالوحدة أو التصنيف، لتكوين التعداد. و لايمكن الإعلان عن تعداد داخل أحد الإجراءات. ولتحديد الترخيص المناسب للوصول إلى الكود، نستخدم كلمات Private، Protected، Friend، أو Public. الأمثلة التالية توضح نماذج من الإعلان عن التعدادات:

```
Private Enum MyEnum
Public Enum MyEnum
Protected Enum MyEnum
Friend Enum MyEnum
Protected Friend Enum MyEnum
```

ومن المفترض أن أول الثوابت فى تعداد تكون قيمته الابتدائية مساوية صفر، والثوابت التالية تتزايد قيمها بمقدار واحد عن القيمة السابقة. على سبيل المثال، تعداد Days التالى، يحتوى على أحد الثوابت الذى يسمى Sunday ذات قيمة قدرها صفر، ثابت باسم Monday وقيمته واحد، وهكذا:

```
Public Enum Days
    Sunday
    Monday
    Tuesday
    Wednesday
    Thursday
    Friday
    Saturday
End Enum
```

ويمكن تخصيص قيم صريحة للثوابت فى تعداد باستخدام عبارة تخصيص. ويجب أن تكون القيمة من أنواع القيم الصحيحة. ويمكن تخصيص قيم سالبة لتمثيل الأخطاء المختلفة. ويمكن الإعلان الصريح عن تعداد بالصيغة التالية:

```
Public Enum MyEnum As Byte
  Zero
  One
  Two
End Enum
```

تأهيل اسماء الثوابت بأسماء التعدادات

عند الإشارة إلى أحد أعضاء تعداد، من المعتاد تأهيل اسم العضو باسم التعداد. على سبيل المثال، للإشارة إلى عضو Sunday فى تعداد Days السابق، نستخدم الصيغة التالية:

```
X = Days.Sunday
```

ويمكن تجنب استخدام تأهيل الأسماء عن طريق استخدام عبارة Imports فى قسم الإعلان بالبرنامج الذى نقوم بكتابته، كما فى الكود التالى:

```
Imports WindowsApplication1.Form1.Days
```

حيث تقوم عبارة Imports باستيراد أسماء مناطق الأسماء (Namespaces) من المشروعات (Projects) ووحدات التجميع (Assemblies) التى يستخدمها التطبيق، ومن داخل نفس التطبيق الذى تظهر به عبارة Imports. وبمجرد إضافة هذه العبارة، يمكن استخدام أعضاء التعداد بدون تأهيل، كما فى الكود التالى:

```
X = Sunday
```

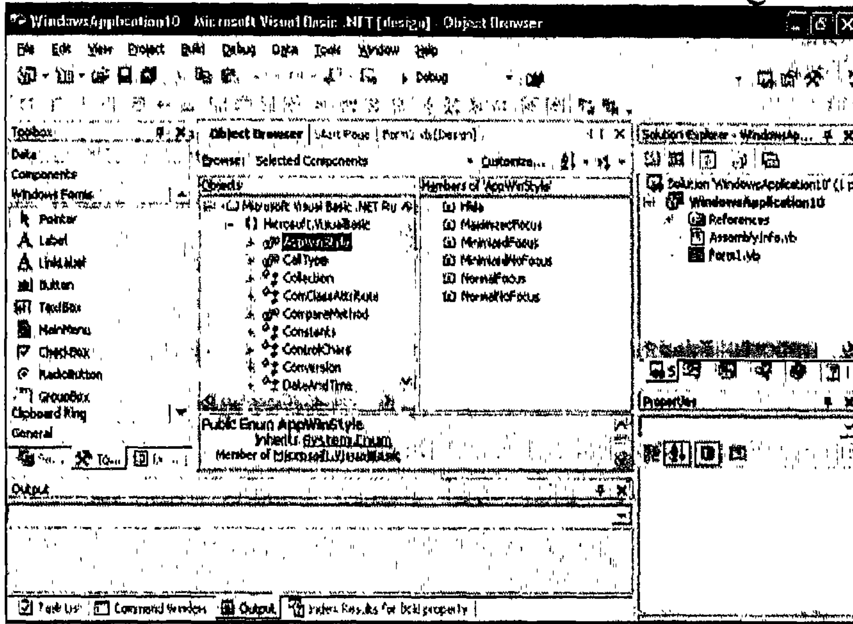
الثوابت والتعدادات الداخلية

يوفر Visual Basic .NET عدد من الثوابت والتعدادات الداخلية لتسهيل عمليات البرمجة. وتعتبر هذه الثوابت أسماء ذات معنى، يتم استخدامها بدلا من الأعداد والسلاسل وتجعل الكود الذى نقوم بكتابته أكثر وضوحا. من ناحية أخرى، تقدم التعدادات الداخلية طريقة سهلة للعمل مع فئات من الثوابت ذات العلاقات. ويعتبر التعداد اسم رمزى لعدد من القيم الثابتة. ويمكن عرض التعدادات والثوابت الداخلية التى يحتوى عليها Visual Basic .NET بمربع Object Browser فى Visual Studio، كما يتضح من الشكل رقم (٧).

المصفوفات (Arrays)

تستخدم المصفوفات للإشارة إلى عدد من المتغيرات التى من نفس النوع، باستخدام

نفس الاسم، ويتم تحديد كل متغير داخل المصفوفة عن طريق رقم هذا المتغير فى فهرس المصفوفة (Index). وتساهم المصفوفات فى اختصار وتبسيط الكود الذى نقوم بكتابته فى حالات كثيرة، بسبب توفر إمكانية معالجة عناصر المصفوفة باستخدام حلقات (Loops) تتعامل بكفاءة مع أى عدد من العناصر.



شكل رقم ٧

المفاهيم المرتبطة باستخدام المصفوفات

يرتبط استخدام المصفوفات ببعض المفاهيم والمصطلحات، مثل أبعاد المصفوفة (Array Dimensions)، حجم المصفوفة (Array Size)، كائن المصفوفة (Array Object)، تصنيف المصفوفة (Array Class)، و أنواع عناصر المصفوفة (Array Element Types).

أبعاد المصفوفة

يمكن أن يكون للمصفوفة بعدا واحدا أو أكثر. ويتساوى عدد الأبعاد مع عدد الأدلة (Subscripts) المستخدمة فى التعرف على كل عنصر من عناصر المصفوفة على حدة. وعدد الأبعاد التى يسمح بها Visual Basic .NET تصل إلى ٣٢ بعدا، غير أنة من المعتاد استخدام أبعاد لا تزيد عن ثلاثة.

حجم المصفوفة

لكل بعد من أبعاد المصفوفة طول يزيد على الصفر. وتتوالى العناصر وراء بعضها فى كل بعد من أبعاد المصفوفة، بدءا من الدليل (Subscript) رقم صفر إلى أعلى رقم دليل فى البعد. وليس للمصفوفة حجم ثابت فى Visual Basic. ولهذا يمكن تغيير حجم المصفوفة بعد تكوينها. وتستخدم عبارة Redim لتخصيص كائن مصفوفة جديدة لمتغير مصفوفة. ولهذا يمكن لعبارة Redim القيام بتغيير طول كل بعد من أبعاد المصفوفة.

المصفوفة التى تتكون من الكائنات

تعتبر المصفوفات من الكائنات فى Visual Basic .NET. ولهذا يعتبر كل نوع من أنواع المصفوفات نوع مرجعي (Reference Type) منفرد. من هذا المفهوم نستطيع استخلاص المضامين التالية:

- متغير المصفوفة يحتوى على مؤشر إلى البيانات التى تكون العناصر ومعلومات الرتبة والطول الخاصة بها.
- عند تخصيص متغير مصفوفة لأخر، يتم فقط نسخ المؤشر.
- لا يعتبر أى اثنين من متغيرات المصفوفات من نفس نوع البيانات إلا إذا كان لهما نفس الرتبة ونفس نوع بيانات العنصر.

تصنيف المصفوفة

كل المصفوفات ترث من تصنيف Array الموجود فى منطقة أسماء System، ويمكن الوصول إلى وسائل وخصائص System.Array باستخدام متغيرات المصفوفات. على سبيل المثال، نستخدم خاصية Rank للحصول على عدد أبعاد التصنيف، ونستخدم وسيلة Sort لفرز عناصر المصفوفة.

نوع عناصر المصفوفة

يحدد الإعلان عن المصفوفة نوع البيانات التى يجب أن تكون جميع عناصر المصفوفة منة. وعندما يكون نوع البيانات Object، يمكن أن يحتوى كل عنصر من عناصر المصفوفة على نوع مختلف من البيانات. ويمكن الإعلان عن المصفوفة باستخدام أى من أنواع البيانات الأساسية، هياكل البيانات، والمصفوفات. ويمكن أيضا الإعلان عن مصفوفات تحتوى على مصفوفات أخرى بعناصرها. فى هذه الحالة، يجب أن تحتوى جميع مصفوفات العناصر على عناصر من نفس نوع البيانات الذى تم تحديده فى الإعلان عن

المصفوفة. وليس ضروريا أن تتساوى أحجام المصفوفات التى تكون المصفوفة الرئيسية.

استخدام المصفوفة

يمكن الإعلان عن مصفوفة للعمل مع فئة من القيم من نفس نوع البيانات. ويمكن النظر إلى المصفوفة على أنها متغير منفرد به كثير من المكونات المستخدمة فى تخزين القيم، بينما تمتلك المتغيرات العادية (Scalar Variables) مكون واحد لتخزين قيمة واحدة. ويمكن استخدام المصفوفة بالكامل عند الحاجة إلى استخدام جميع البيانات التى تحتوى عليها، كما يمكن استخدام العناصر المنفردة على أساس واحد فى كل مرة. على سبيل المثال، لتخزين المبيعات الشهرية لسنة كاملة، يمكن تكوين مصفوفة من اثنى عشر عنصرا بدلا من تكون اثني عشر متغيرا مستقلا. وكل عنصر فى المصفوفة يحتوى على قيمة واحدة يمكن الوصول إليها بتحديد فهرس العنصر. المثال التالى، يعلن عن مصفوفة تتكون من أثني عشر عنصرا من نوع Decimal، ثم يقوم بتخزين القيمة ٢٠ فى كل عنصر:

```
Sub FillArray ()
  Dim CurExpense (12) As Decimal
  Dim I As Integer
  For I = 0 to 12
    CurExpense (I) = 20.00
  Next I
End Sub
```

ويمكن تنويع أنواع البيانات فى مصفوفة عن طريق إعلانها من نوع Object، كما يتضح من المثال التالى:

```
Dim EmployeeData (3) As Object
EmployeeData (0) = "Ron Bendel"
EmployeeData (1) = "4242 Maple Blvd"
EmployeeData (2) = 48
EmployeeData (3) = "06-09-1953"
```

المصفوفات متعددة الأبعاد

فى Visual Basic، يمكن الإعلان عن مصفوفات تحتوى على أبعاد بحد أقصى ٣٢. على سبيل المثال، العبارة التالية تعلن عن مصفوفة من بعدين، أحدهما خمسة صفوف والأخر عشرة أعمدة.

```
Dim Rectangle (4, 9) As Single
```

ويساوى عدد العناصر فى مصفوفة مجموع أعداد عناصر الأبعاد المختلفة، الذى يساوى

فى هذه الحالة ٥٠ (١٠ × ٥). ويتم الحصول على رتبة المصفوفة، التى تمثل عدد الأبعاد بها، باستخدام خاصية Rank بمتغير المصفوفة. كما نستخدم وسيلة GetLength للحصول على طول كل بعد من أبعاد المصفوفة. ومن المعتاد أن يكون أقل قيمة دليل فى أى بعد من أبعاد المصفوفة تساوى صفر، بينما يمكن الحصول على أعلى قيمة فى البعد باستخدام وسيلة GetUpperBound. كما يجب ملاحظة أن عدد أبعاد المصفوفة يبدأ بالرقم صفر أيضا.

ويمكن استخدام حلقات For المتداخلة لمعالجة المصفوفات المتعددة الأبعاد بكفاءة. على سبيل المثال، يقوم الكود التالى بجعل القيم الابتدائية لعناصر التصنيف MatrixA تتراوح بين القيمة صفر والقيمة ٩٩، على أساس موقع العنصر فى المصفوفة:

```
Dim I, J As Integer
Dim MaxDim0, MaxDim1 As Integer
Dim MatrixA(9, 9) As Double
MaxDim0 = MatrixA.GetUpperBound (0)
MaxDim1 = MatrixA.GetUpperBound (1)
For I = 0 To MaxDim0
  For J = 0 To MaxDim1
    MatrixA (I, J) = (I * 10) + J
  Next J
Next I
```

الإعلان عن متغيرات المصفوفة

يتم الإعلان عن متغيرات المصفوفة بنفس الطريقة التى يتم بها الإعلان عن باقى المتغيرات، باستخدام عبارة Dim. ويجب أن يتبع اسم المصفوفة زوج أو أكثر من الأقواس للإشارة إلى أن المتغير يمثل مصفوفة وليس متغيرا يحتوى على قيمة واحدة. وفيما يلى، نعرض الصيغ المختلفة للإعلان عن متغيرات المصفوفات:

للإعلان عن متغير مصفوفة ذات بعد واحد بدون تخصيص ذاكرة له، نستخدم صيغة الكود التالية:

```
Dim MySingleArray () As Integer
```

وللإعـلان عـن متغير مصفوفة من أربعة أبعاد بدون تخصيص ذاكرة له، نستخدم صيغة الكود التالية:

```
Dim My4DArray (,,,) As Short
```

وللإعلان عن متغير مصفوفة يحتوى على مصفوفات (Jagged Array Variable) من

نوع Byte بدون تخصيص ذاكرة، نستخدم صيغة الكود التالية:

```
Dim MyJaggedArray ()()() As Byte
```

ولتخصيص ذاكرة لعناصر المصفوفة، يجب تحديد عدد أو قيم عناصر المصفوفة عند الإعلان عن المتغير ، كما يتضح من الكود التالى:

```
Dim MyArray (5) As Integer
Dim MyArray () As Integer = {0, 0, 0, 0, 0}
```

وفى حالة عدم تحديد عدد أو قيم العناصر عند الإعلان عن متغير المصفوفة، يجب تخصيص ذاكرة لمتغير المصفوفة قبل تخزين بيانات بها. لتنفيذ ذلك، نقوم بتخصيص كائن مصفوفة للمتغير باستخدام كلمة New، كما يتضح من الكود التالى:

```
Variablename = New DataType ([length]) {[values]}
```

إعداد المصفوفات

يمكن إعداد (Initializing) متغير مصفوفة على أساس أنه جزء من الإعلان عن المتغير. ويمكن تنفيذ واحد من المهام التالية فى عبارة الإعلان:

- تحديد الطول الإبتدائى لواحد أو أكثر من أبعاد المصفوفة فى الأقواس التى تلى اسم المتغير، بدون تخصيص كائن مصفوفة للمتغير.
- تخصيص كائن مصفوفة للمتغير، باستخدام فقرة New. وعند استخدام فقرة New، يجب أن يتبعها رمز ({})، حتى فى حالة خلوهما من أى بيانات.
- تخصيص كائن مصفوفة للمتغير وإدخال الأطوال الابتدائية للأبعاد فى فقرة New.
- تخصيص كائن مصفوفة للمتغير وإدخال قيم ابتدائية للعناصر فى فقرة New.

وعند تحديد أطوال الأبعاد فى الأقواس التى تلى اسم المتغير، يجب استخدام عبارة تخصيص بعد ذلك لتخصيص قيم لعناصر المصفوفة. تبين نماذج الإعلانات التالية كيفية إعداد متغير مصفوفة ذات بعد واحد:

- الإعلان عن متغير مصفوفة يحتوى على ثلاثة عناصر من نوع Byte.

```
Dim BA(2) As Byte
```

- الإعلان عن متغير يحتوى على كائن مصفوفة خالية.

```
Dim BA() As Byte = New Byte() {}
```

- الإعلان عن متغير يحتوى على القيم من صفر إلى ٢.

```
Dim BA() As Byte = New Byte() {0,1,2}
```

- الإعلان عن متغير مع تحديد الطول الإبتدائى للبعد والقيم الابتدائية للعناصر.

```
Dim BA() As Byte = New Byte(2) {0,1,2}
```

ويمكن إعداد متغير مصفوفة متعددة الإبعاد بنفس الطريقة، كما يتضح من الكود التالى:

- الإعلان عن متغير مصفوفة ذات بعدين، كل منهما يحتوى على عنصرين.

```
Dim SA(1, 1) As Short
```

- الإعلان عن متغير مصفوفة خالية.

```
Dim SA(,) As Short = New Short(,) {}
```

- الإعلان عن متغير مصفوفة ذات بعدين، يحتوى كل منهما على عنصرين.

```
Dim SA(,) As Short = New Short(1, 1) {}
```

- الإعلان عن متغير مصفوفة ذات بعدين، كل منهما يحتوى على عنصرين وتحتوى على أربعة قيم.

```
Dim SA(,) As Short = New Short(,) {{5, 6}, {7, 8}}
```

وعند الإعلان عن متغير مصفوفة تحتوى على مصفوفات أخرى، يمكننا تحديد أطوال أبعاد المصفوفة الأولى فقط. الكود التالى يعرض أمثلة عن كيفية إعداد متغيرات مصفوفات تحتوى على مصفوفات:

- الإعلان عن متغير مصفوفة تتكون من مصفوفتين.

```
Dim JB(1)() As Byte
```

- الإعلان عن متغير مصفوفة من مصفوفات خالية.

```
Dim JB()() As Byte = {New Byte() {}, New Byte() {}}
```

- الإعلان من متغير مصفوفة من مصفوفات بها قيم افتراضية.

```
Dim JB()() As Byte = {New Byte(1) {}, New Byte(1) {}}
```

- الإعلان عن متغير مصفوفة من مصفوفتين، تحتوى كل منهما على قيمتين.

```
Dim JB()() As Byte = {New Byte() {5, 6}, New Byte() {7, 8}}
```

تغيير أحجام المصفوفات

يمكن تغيير حجم مصفوفة فى أى وقت عن طريق تخصيص كائن مصفوفة مختلف لنفس متغير المصفوفة باستخدام عبارة ReDim أو عبارة تخصيص عادية. ويمكن لكائن المصفوفة الجديد أن يحتوى على أبعاد مختلفة ورتبة مختلفة، مما يساعد فى إدارة الذاكرة بكفاءة. على سبيل المثال، قد نحتاج إلى استخدام مصفوفة طويلة لمدة قصيرة، مما يترتب

علية ضرورة إعادة تقصير حجمها باستخدام عبارة ReDim للاستفادة من الذاكرة غير المستخدمة.

عند إعادة تحديد حجم إحدى الصفوفات باستخدام Redim، تفقد المصفوفة القيم التى تحتفظ بها. ويمكن الحفاظ على هذه القيم باستخدام الكلمة المرشدة Preserve فى عبارة Redim. على سبيل المثال، العبارة التالية تحدد مصفوفة جديدة، تقوم بالإعداد الإبتدائى لعناصرها باستخدام عناصر مصفوفة قائمة هى MyArray، ثم تخصص المصفوفة الجديدة للمتغير MyArray:

```
ReDim Preserve MyArray(10, 20)
```

وفيما يختص بالمصفوفة المتعددة الأبعاد، يمكننا تغيير البعد الأخير فقط عند استخدام Preserve. فإذا حاولنا تغيير أى بعد أخر فى المصفوفة، فسوف نحصل على رسالة خطأ. وعندما لا نعرف الحجم الحالى للبعد، نستخدم وسيلة GetUpperBound، التى تعيد إلينا أعلى دليل بالبعد الذى نحدده. فى المثال التالى، يكون السطر الأول صحيحا بينما يكون السطر الثانى غير صحيح بسبب محاولته تغيير أول البعدين فى المصفوفة.

```
ReDim Preserve Matrix(Matrix.GetUpperBound(0), Matrix.GetUpperBound(1) + 10)
ReDim Preserve Matrix(Matrix.GetUpperBound(0) + 10, Matrix.GetUpperBound(1))
```

السمات المتقدمة للمصفوفات

يمكن تخصيص محتويات أحد المصفوفات لمصفوفة أخرى، تكوين دوال (Functions) تقبل وتعيد مصفوفات، تكوين خصائص (Properties) تقبل وتعيد مصفوفات.

تخصيص المصفوفات

بالنظر إلى إن المصفوفة تعتبر من الكائنات، لهذا يمكن استخدامها فى عبارات التخصيص مثل أنواع الكائنات الأخرى. وعند القيام بتخصيص مصفوفة لأخرى، تنطبق القواعد التالية:

- يجب أن يكون عدد أبعاد مصفوفة المقصد مساويا لعدد أبعاد مصفوفة المصدر.
- عند تساوى عدد الأبعاد فى المصفوفتين، ليس من الضروري أن تتساوى أطوال هذه الأبعاد.

- يجب أن تحتوى عناصر كل من المصفوفتين على قيم (Values) أو على مراجع (References) فقط.
- عندما تحتوى عناصر المصفوفتين على قيم، يجب أن تكون هذه القيم من نفس النوع.
- عندما تحتوى عناصر المصفوفتين على مراجع، يجب أن يكون هناك تحويل توسيع (Widening Conversion) من عناصر المصدر إلى عناصر المقصد.

المثال التالى، يوضح تخصيص مصفوفة لمصفوفة أخرى تساويها فى عدد الأبعاد وفى نوع البيانات، ولا تساويها فى طول البعد:

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
        Dim MyArray() As Integer = {10, 20, 30, 40, 50, 60}
        Dim YourArray(4) As Integer
        YourArray = MyArray
        Debug.WriteLine(YourArray(1))
    End Sub
```

إعادة مصفوفة من دالة

يمكن لإجراء Function أن يعيد مصفوفة من القيم، كما يتضح من المثال التالى، الذى تقوم فيه دالة ArrayFunction باستقبال قيمة من نوع Byte وتعيد مصفوفة من نوع Byte أيضا.

```
Dim B As Byte = CByte(54)
Dim I As Integer
Dim ReturnArray() As Byte
ReturnArray = ArrayFunction(B)
For I = 0 To ReturnArray.GetUpperBound(0)
  Debug.WriteLine("Byte " & I & ": " & ReturnArray(I))
Next I

Public Function ArrayFunction(ByVal B As Byte) As Byte()
  Dim X(2) As Byte
  X(0) = B
  X(1) = B + B
  X(2) = B + CByte(200)
  Return X
End Function
```

بعد تشغيل هذا المثال، سوف يحتوى المتغير ReturnArray على مؤشر إلى مصفوفة من

ثلاثة عناصر تحتوى على قيم العناصر التى تم تخصيصها للمصفوفة X فى دالة ArrayFunction. فى هذه الحالة، يجب أن تكون عناصر مصفوفة ReturnArray من نفس نوع العناصر التى تعيدها الدالة لأنها تحتوى على قيم.

أنواع البيانات

يحتوى Visual Basic .NET على عدد من أنواع البيانات الأساسية (Elementary Data Types). كما يمكن تكوين أنواع بيانات خاصة بنا (Composite Data Types)، تتكون من أنواع البيانات الأساسية، مثل هياكل البيانات (Structures) والتصنيفات (Classes).

وتنقسم أنواع البيانات الأساسية إلى المجموعات التالية:

- البيانات العددية (Numeric)، سواء كانت أعداذ صحيحة أو أعداد بها كسور عشرية.
- بيانات الحروف (Character) ، سواء كانت حروف منفصلة أو مجموعة من الحروف معا.
- بيانات متنوعة (Miscellaneous) ، مثل البيانات المنطقية (Boolean)، والتاريخ (Dates).

وتختلف هذه الأنواع فى المساحة التى تشغلها فى الذاكرة، كما تختلف فى نطاق القيم التى يمكن تخزينها بها. الجدول رقم (٧) يعرض ملخصا بالأنواع الأساسية، القيم، ومساحات التخزين المطلوبة لكل نوع.

النوع	الحجم	نطاق القيمة
Boolean	٢	القيمة True أو القيمة False
Byte	١	من 0 إلى 255
Char	٢	من 0 إلى 65535
Date	٨	من ا يناير ، ٠٠٠١ إلى ٣١ ديسمبر ، ٩٩٩٩
Decimal	١٢	+/-79,228,162,514,264,337,593,543,950,335 بدون العلامة العشرية

نطاق القيمة	الحجم	النوع
+/-7.9228162514264337593543950335 مع ٢٨ كسر عشرى		
-1.79769313486231E308 إلى -4.94065645841247E-324 للقيم السالبة 4.94065645841247E-324 إلى 1.79769313486232E308 للقيم الموجبة	٨	Double
-2,147,483,698 إلى 2,147,483,647	٤	Integer
-9,223,372,036,854,775,808 إلى 9,223,372,036,854,775,807	٨	Long
يمكن تخزين أى نوع فى متغير من هذا النوع	٤	Object
-32,768 إلى 32,767	٢	Short
-3,402823E38 إلى -1.401298E-45 للقيم السالبة 1.401298E-45 إلى 3.402823E38 للقيم الموجبة	٤	Single

جدول ٧

من الملاحظ فى الجدول رقم (٧) أن كل نوع من أنواع البيانات يتطلب عدد محدد من وحدات البايت لتخزين نطاق البيانات التى يمكنة الاحتفاظ بها. وحدة البايت هى أصغر وحدة معلومات يستطيع الكمبيوتر معالجتها. وتتكون وحدة البايت من ثمانية أجزاء، يطلق على كل منها bit. وتكون أحجام أنواع البيانات مساوية للبايت أو مضاعفاته. ولا يمكن تغيير عدد وحدات البايت المطلوبة لتخزين أحد أنواع البيانات. كما أن تحديد هذا العدد يتم بواسطة نظام التشغيل المستخدم. ويجب معرفة عدد وحدات البايت التى يتكون منها كل نوع من أنواع البيانات لكى يمكن اختيار النوع المناسب لمهام البرمجة التى نقوم بها. ويتحكم عدد وحدات البايت التى يتكون منها النوع فى نطاق القيم التى يمكن تخزينها بدءا من أصغر قيمة إلى أكبر قيمة. على سبيل المثال، نطاق القيم التى يمكن تخزينها فى

واحد بايت يبدأ من صفر إلى ٢٥٥. يعنى ذلك، أن عدد القيم الممكن تخزيها فى البايت الواحد يساوى ٢٥٦ قيمة. وبالمثل، يمكن لنوع البيانات Integer أن يحتوى على ٤٢٩٤٩٦٧٢٩٦ قيمة لأنة يتكون من أربع وحدات بايت. تبدأ هذه القيم من — ٢١٤٧٤٨٣٦٤٨ إلى ٢١٤٧٤٨٣٦٤٧. ويعتبر نطاق البيانات التى يمكن إدخالها فى نوع البيانات ذات أهمية كبيرة، بالنظر إلى أن إدخال قيم أقل أو أكبر من هذا النطاق قد يترتب علية فشل البرنامج وإطلاق أخطاء.

انواع البيانات العددية

يمكن تقسيم البيانات العددية (Numeric Data Types) إلى مجموعتين من أنواع البيانات. المجموعة الأولى هى مجموعة أنواع الأعداد الصحيحة (Integral Types)، التى تشمل الأعداد الصحيحة. والمجموعة الثانية هى مجموعة أعداد العلامة العشرية المتحركة (Floating Point Types)، التى تشمل الأعداد التى تحتوى على أعداد صحيحة وكسر عشرى. الجدول رقم (٨) يحتوى على أنواع البيانات العددية المختلفة، عدد وحدات البايت التى تستخدمها، ما إذا كان النوع يحتوى على إشارة، ونطاق الأعداد التى يمكن تخزينها باستخدام ذلك النوع.

النوع	الحجم	الإشارة	نطاق القيمة
Byte	١	لا يوجد	من 0 إلى 255
Decimal	١٢	يوجد	+/- 79,228,162,514,264,337,593,543,950,335 بدون العلامة العشرية +/-7.9228162514264337593543950335 مع ٢٨ كسر عشرى
Double	٨	يوجد	-1.79769313486231E308 إلى -4.94065645841247E-324 للقيم السالبة 4.94065645841247E-324 إلى

النوع	الحجم	الإشارة	نطاق القيمة
			1.79769313486232E308 للقيم الموجبة
Integer	٤	يوجد	-2,147,483,698 إلى 2,147,483,647
Long	٨	يوجد	-9,223,372,036,854,775,808 إلى 9,223,372,036,854,775,807
Short	٢	يوجد	-32,768 إلى 32,767
Single	٤	يوجد	-3,402823E38 إلى -1.401298E-45 للقيم السالبة 1.401298E-45 إلى 3.402823E38 للقيم الموجبة

جدول ٨

اكبر قيمة موجبة يمكن تخزينها فى صورة رقم ليس به إشارات، هى ٢ مرفوعة لأس يساوى عدد وحدات بت فى نوع بياناتها ناقصا واحد. على سبيل المثال، أكبر قيمة صحيحة يمكن تخزينها فى نوع Byte تساوى (٢ ^ ٨) - ١ أو ٢٥٥. ويرجع السبب فى خصم واحد من القيمة، إلى أن عدد القيم التى يمكن أن يحتويها هذا النوع يساوى ٢٥٦ واحدى هذه القيم هو الصفر. ويترتب على وجود إشارة بالنوع الرقمى تخفيض أعلى قيمة موجبة به عن طريق تخفيض الأس المستخدم فى الصيغة السابقة بمقدار واحد. على سبيل المثال، نوع البيانات Short يحتوى على عدد ٢ بايت أو ١٦ بت، ولذلك تكون أعلى قيمة موجبة به تساوى (٢ ^ ١٥) — ١، لأن الإشارة تستخدم واحد بت من حجم النوع أو (١٦-١). من ناحية أخرى، يبقى عدد القيم التى يمكن أن يحتويها النوع ثابتا بغض النظر عن وجود أو عدم وجود الإشارة.

أنواع الأعداد الصحيحة

تمثل أنواع الأعداد الصحيحة (Integral Types) الأعداد الصحيحة فقط، لذلك لا يمكنها الاحتفاظ بالكسور. وعند محاولة تخصيص رقم به كسر لنوع من هذه الأنواع، فإن

الكسور يجرى تقريبها إلى أعداد صحيحة. غير أن هذا التقريب قد يترتب علية الحصول على نتائج غير صحيحة للعمليات الحسابية. يعرض الجدول رقم (٩) الأنواع الخاصة بالأعداد الصحيحة.

النوع	الحجم	الإشارة	نطاق القيمة
Byte	١	لا يوجد	من 0 إلى 255
Integer	٤	يوجد	-2,147,483,698 إلى 2,147,483,647
Long	٨	يوجد	-9,223,372,036,854,775,808 إلى 9,223,372,036,854,775,807
Short	٢	يوجد	-32,768 إلى 32,767

جدول ٩

أنواع أعداد العلامة العشرية المتحركة

ترتبط أنواع أعداد العلامة العشرية المتحركة (Floating Point Types) بالأعداد التى تحتوى على جزأين. الجزء الأول رقم صحيح والجزء الثانى كسر عشرى. وتحتوى جميع أنواع العلامة العشرية المتحركة على إشارة تحديد الموجب والسالب. تشتمل هذه الأنواع على نوع Decimal، الذى يحتوى على عدد ٢٨ رقم على يمين العلامة العشرية. ويناسب هذا النوع العمليات المالية التى تتطلب عدد كبير من أرقام الكسور لأنها لا تستطيع التغلب على مشاكل التقريب.

كما تشمل هذه الأنواع Single و Double، التى تشتمل على نطاق بيانات أكبر من نطاق Decimal، ولكن مع عدد أقل من الأرقام على يمين العلامة العشرية. يدعم نوع Single عدد ٧ أرقام على يمين العلامة العشرية، بينما يدعم نوع Double عدد ١٥ رقم على يمين العلامة العشرية. ولتوضيح هذه الأنواع، يعرض الجدول رقم (١٠) أسماء الأنواع ، أحجامها، ونطاق البيانات التى يمكن الاحتفاظ بها. ولا يعرض الجدول عمودا لإشارة الموجب والسالب، لأن جميع هذه الأنواع تحتوى على إشارة.

النوع	الحجم	نطاق القيمة
Decimal	١٢	+/-79,228,162,514,264,337,593,543,950,335 بدون العلامة العشرية +/-7.9228162514264337593543950335 مع ٢٨ كسر عشرى
Double	٨	-1.79769313486231E308 إلى -4.94065645841247E-324 للقيم السالبة 4.94065645841247E-324 إلى 1.79769313486232E308 للقيم الموجبة
Single	٤	-3,402823E38 إلى -1.401298E-45 للقيم السالبة 1.401298E-45 إلى 3.402823E38 للقيم الموجبة

جدول ١٠

نوع بيانات الرمز

قيم المتغيرات من نوع Char يتم الاحتفاظ بها فى صورة أرقام بدون إشارات (Unsigned)، طول كل منها ٢ بايت، ويبدأ نطاق هذه الأعداد من الصفر إلى ٦٥٥٣٥. كل عدد من هذه الأعداد يمثل رمز أحادى الكود (Unicode Character). والتحويل المباشر بين نوع Char وبين الأنواع العددية غير ممكن، ولكن يمكننا تحقيق ذلك باستخدام دالة AscW و دالة ChrW. وعند تحويل سلسلة رموز إلى نوع Char، يجب أن تحتوى السلسلة على رمز واحد، كما يجب عدم تفعيل عبارة Option Strict. وفى حالة تفعيل عبارة Option Strict، يجب إلحاق حرف C بسلسلة الرموز للتعريف بأنها من نوع Char، كما يتضح من المثل التالى :

```
Option Strict
Dim h As Char
h = "F"c
```

نوع بيانات السلسلة

يمكن تعريف السلسلة (String) بأنها مجموعة تتكون من واحد أو أكثر من رموز Unicode. ويشغل كل رمز من رموز السلسلة موقعا فى الذاكرة يحتوى على اثنين من وحدات البايت، ويمكن أن يحتوى هذا الموقع على قيمة تتراوح من صفر إلى ٦٥٥٣٥.

ودائما يكون المتغير أو المعامل من نوع السلسلة متغير الطول، حيث يتمدد أو ينكمش عند تخصيص بيانات جديدة له. ويحتوى نوع سلسلة الرموز على دوال داخلية تمكننا من تنفيذ الكثير من المعالجات على السلسلة، وتماثل هذه المعالجات تلك التى تسمح بها المصفوفات.

معالجة السلسلة

يوفر لنا تصنيف String فى NET Framework.، الكثير من الوسائل لتسهيل عمليات مقارنة ومعالجة السلاسل. كما تحتوى لغة Visual Basic .NET أيضا، على مجموعة من الوسائل الموروثة تماثل الكثير من الوسائل الموجودة فى تصنيف String. ويمكن استخدام الوسائل التى يوفرها Visual Basic بدون تأهيل، كما يتضح من الكود التالى:

```
Dim aString As String = "SomeString"
Dim bString As String
bString = Mid(aString, 3, 3)
```

ويمكن أيضا استخدام الوسائل المتوفرة فى تصنيف String لمعالجة السلاسل. ويمكن تقسم الوسائل المتوفرة فى تصنيف String إلى نوعين: وسائل مشتركة (Shared Methods)، ووسائل الكائنات (Instances Methods). الوسائل المشتركة هى التى يمكن استخدامها بعد تأهيلها باسم التصنيف، بينما وسائل الكائنات يتم استخدامها بعد تأهيلها باسم الكائن. الكود التالى، يوضح نماذج من كلا النوعين:

```
Dim aString As String = "A String"
Dim bString As String
bString = String.Copy("A literal string")
bString = aString.SubString(2,6) ' bString = "String"
```

فى عبارة التخصيص الأولى بالكود السابق، تم استخدام وسيلة Copy بعد تأهيلها باسم التصنيف، ولذلك تعتبر وسيلة مشتركة بين جميع أمثلة تصنيف String. وفى عبارة التخصيص الثانية، تم تأهيل وسيلة SubString باسم مثل التصنيف، وهو فى هذه الحالة aString.

مقارنة السلاسل

يمكن المقارنة بين السلاسل باستخدام وسيلة String.Compare الموجودة فى تصنيف String. كما يمكن المقارنة أيضا باستخدام دالة StrComp فى Visual Basic. وتتم المقارنة بين السلاسل فى هذه الحالة على أساس ترتيب الحروف داخل كل سلسلة. يوضح الكود التالى استخدام وسيلة String.Compare للمقارنة بين سلسلتين:

```
Dim myString As String = "Alphabetical"
Dim secondString As String = "Order"
Dim result As Integer
result = String.Compare (myString, secondString)
```

تعيد هذه الوسيلة قيمة صحيحة تعبر عن العلاقة بين السلسلتين. إذا كانت القيمة العائدة موجبة، فإن ذلك يعنى أن السلسلة الأولى أكبر من السلسلة الثانية، والعكس صحيح عندما تكون القيمة سالبة. وتكون القيمة العائدة صفر عند تساوى السلسلتين.

البحث فى السلاسل

يمكن النظر إلى السلسلة على أنها مصفوفة من الرموز، وبالتالى يمكن الحصول على أحد الرموز باستخدام فهرس الرمز داخل السلسلة من خلال خاصية Chars، كما يتضح من المثال التالى:

```
Dim myString As String = "ABCDE"
Dim myChar As Char
myChar = myString.Chars(3)
```

ويمكن استخدام وسيلة String.IndexOf للحصول على فهرس أحد الرموز فى سلسلة، كما فى المثال التالى:

```
Dim myString As String = "ABCDE"
Dim myInteger As Integer
myInteger = myString.IndexOf("D")
```

وللحصول على أخر موقع يوجد به أحد الحروف بسلسلة، نستخدم وسيلة String.LastIndexOf، كما يتضح من الكود التالى:

```
Dim myString As String = "ABCDEAN"
Dim myInteger As Integer
myInteger = myString.LastIndexOf("A")
Debug.WriteLine(myInteger)
```

تكوين سلاسل جديدة من سلاسل قائمة

يقدم تصنيف System.String مجموعة خيارات واسعة لتعديل، معالجة، وتكوين سلاسل جديدة من سلاسل قديمة. لتجميع عدة سلاسل معا، نستخدم عوامل ربط السلاسل، التى تشمل (& و +). ويمكن أيضا استخدام وسيلة String.Concat لربط السلاسل، كما يتضح من الكود التالى:

```
Dim aString As String = "A"
Dim bString As String = "B"
```

```
Dim myString As String
myString = String.Concat(aString, bString)
```

ولتحويل السلاسل إلى الحروف الكبيرة (Uppercase) أو الحروف الصغيرة (Lowercase)، نستخدم دوال Ucase و Lcase الموجودة فى Visual Basic .NET أو وسيلة String.ToUpper ووسيلة String.ToLower الموجودة فى تصنيف String. وتقوم وسيلة String.Fromat فى تصنيف String و أمر Format فى Visual Basic بتوليد سلسلة جديدة عن طريق تطبيق صياغة جديدة لسلسلة موجودة. ولحذف الفراغات الموجودة فى بداية أو نهاية السلسلة، يمكننا استخدام وسيلة String.Trim فى تصنيف String أو دالة Trim فى Visual Basic. كما يمكن حذف الفراغات التى على يسار السلسلة أو على يمينها باستخدام وسيلة String.TrimStart ووسيلة String.TrimEnd فى تصنيف String . ويمكن أيضا استخدام دالة Ltrim و دالة Rtrim فى Visual Basic .NET. وتستخدم وسائل String.PadLeft وString.PadRight لإضافة رموز فى بداية السلسلة أو فى نهايتها. وتستخدم وسيلة System.Remove لإزالة حروف من جسم السلسلة. وتقوم وسيلة String.Replace بإحلال حروف فى جسم السلسلة، كما يتضح من الكود التالى:

```
myString = aString.Remove(14, 5)
anotherString = myString.Replace("My", "Another")
```

ونستخدم وسيلة String.Insert لإدراج حروف فى سلسلة، كما فى الكود التالى:

```
myString = aString.Insert(13, "ri")
```

يمثل المعامل الأول فى وسيلة String.Remove أو String.Insert، فهرس الحرف الذى يبدأ منة الحذف أو الإضافة، ويمثل المعامل الثانى فى وسيلة Remove طول السلسلة التى يتم حذفها، وفى وسيلة Insert السلسلة التى يتم إدراجها. ويمكن ربط سلاسل مصفوفة باستخدام سلسلة أخرى للفصل بينها. للقيام بذلك، نستخدم وسيلة String.Join، التى يوضحها المثال التالى:

```
shoppingList = String.Join(",", shoppingItem)
```

فى هذا المثال، يمثل المتغير shoppingList المصفوفة الناتجة، ويمثل المتغير shoppingItem مصفوفة تحتوى على سلاسل الرموز التى نريد الربط بينها باستخدام سلسلة الرمز ",". وعلى العكس من الوسيلة السابقة، يمكن استخدام وسيلة String.Split لتحويل سلسلة إلى مصفوفة من السلاسل، كما يتضح من المثال التالى:

```
shoppingItem = shoppingList.Split(","c)
```

ولتقسيم سلسلة إلى مجموعة من السلاسل الفرعية، نستخدم وسيلة String.Substring، كما فى الكود التالى:

```
subString = aString.SubString(5,6)
```

المعامل الأول فى وسيلة SubString يمثل الفهرس الذى تبدأ عندة السلسلة الفرعية، والمعامل الثانى يمثل طول السلسلة الفرعية.

انواع بيانات متنوعة

هناك عدد أخر من أنواع البيانات التى تمثل جزءا من البيانات الأساسية بالنظام، تشمل البيانات المنطقية (Boolean)، التاريخ (Date)، والكائن (Object). يتم تخزين القيمة المنطقية فى صورة رقم طولة ١٦ بت، ويمكن أن تحتوى على واحدة من قيمتين، قيمة حقيقية (True) أو قيمة غير حقيقية (False). وتستخدم الكلمات المفتاحية True و False لتخصيص قيمة لمتغير منطقي. ويتم تخزين القيم المنطقية فى صورة أعداد، حيث يعتبر الصفر False، و أى رقم غير الصفر يعتبر True. ويتم تخزين قيمة True داخليا فى صورة (-1).

ويمثل المتغير من نوع Date، تاريخ ووقت بين الساعة ١٢ صباحا بتاريخ الأول من يناير سنة واحد إلى الساعة ٢٣:٥٩:٥٩ بتاريخ ٣١ ديسمبر سنة ٩٩٩٩. ويتم تخزين التواريخ داخليا فى صورة أرقام صحيحة طويلة (Long Integers)، و كل زيادة تمثل ١٠٠ جزء من الثانية من الوقت المنقضى منذ الساعة ١٢ صباحا بتاريخ الأول من يناير سنة واحد. ويجب وضع الرموز التى تكون التاريخ بين علامتين من علامات الأعداد (#)، ويجب أن يكون فى صيغة m/d/yyyy، كما يوضحه الكود التالى:

```
Private SomeDate As Date = #5/4/2002 7:48 AM"
```

ويمثل نوع البيانات Object نوع الأساس فى Visual Basic .NET. يعنى ذلك أن كل الأنواع الأخرى مشتقة من هذا النوع. ويعنى ذلك أيضا أن أى نوع بيانات أخر يمكن تحويلة إلى نوع Object. ويطلق على Object النوع الفضفاض بسبب إمكانية استخدامه لتخزين جميع أنواع البيانات الأخرى. لتوضيح ذلك، نعرض الكود التالى:

```
Dim V As Object
V = "17"
V = V – 15
```

هياكل البيانات

يمكن تجميع عناصر بيانات ذات أنواع مختلفة فى هيكل بيانات (Structures). وبعد الإعلان عن هيكل البيانات، يصبح من أنواع البيانات المركبة (Composite Data Type)، وبالتالى يمكن استخدامه فى تحديد نوع بيانات المتغيرات التى نقوم بالإعلان عنها. ويمكن أن يحتوى هيكل البيانات على حقول (Fields)، خصائص (Properties)، وسائل (Methods)، وأحداث (Events). ويمكن لهيكل البيانات أن يقوم بتنفيذ واجهة بينية (Interface) أو أكثر. وعند الإعلان عن أعضاء هيكل البيانات، يمكن استخدام التراخيص (Accessibility) المختلفة. وتظهر فائدة هيكل البيانات عندما نريد لمتغير واحد أن يحتفظ بأجزاء متعددة من البيانات ذات الأنواع المختلفة والعلاقات البينية.

الإعلان عن هيكل بيانات

نبدأ الإعلان عن هيكل بيانات باستخدام عبارة Structure، و ينتهى الإعلان بعبارة End Structure. وبين هاتين العبارتين نعلن عن أعضاء الهيكل، التى يجب أن تحتوى على عضو واحد على الأقل. ويمكن استخدام أى نوع من أنوع البيانات فى الإعلان عن أعضاء الهيكل. على سبيل المثال، يمكن الإعلان عن هيكل بيانات يحتوى على بيانات الموظف بالصيغة التالية:

```
Structure Employee
   Public GivenName As String
   Public FamilyName As String
   Public Extension As Long
   Private Salary As Decimal
End Structure
```

ويمكن تحديد ترخيص الوصول إلى الهيكل باستخدام كلمات Public، Protected،

Friend، أو Private، ويمكن قبول الترخيص الإفتراضى، وهو Public، كما هو. وبالإضافة إلى تحديد ترخيص الوصول إلى الهيكل عموما، يجب تحديد ترخيص الوصول إلى كل عضو من أعضائه عند الإعلان عنها. ويصبح الوصول عام (Public) عند استخدام عبارة Dim بدون أى كلمات مرشدة للإعلان عن أعضاء هيكل البيانات. ويجب ملاحظة أننا لا نستطيع إعداد أى عضو من أعضاء الهيكل عند الإعلان عنة، ويتم ذلك بعد الإعلان عن متغير من نوع الهيكل.

متغيرات الهياكل

بعد الإعـلان عن هيكل بيانات، يمكن الإعلان عن متغيرات من نوع ذلك الهيكل على مسـتوى الوحـدة أو الإجراء. على سبيل المثال، يمكن تكوين هيكل بيانات يسجل معلومات دليل حسابات، كما فى الكود التالى:

```
Public Structure Acchart
  Public AcID As String
  Public AcName As String
  Public AcBal As Decimal
End Structure
```

يمكن بعد ذلك الإعلان عن متغيرات من هذا النوع، كما يتضح من الصيغة التالية:

```
Dim MyChart As Acchart
```

الوصول إلي القيم فى هياكل البيانات

لتخصـيص قيم لعناصر متغير هيكل بيانات واسترجاعها منها، نستخدم نفس التركيب اللغـوى المستخدم مع الكائنات. على سبيل المثال، يمكن الوصول إلى عناصر متغير MyChart السابق الإعلان عنة، باستخدام الكود التالى:

```
MyChart.AcID = "100"
MyChart.AcName = "xxxxxxx"
MyChart.AcBal = 520.25
```

ويمكن أيضا تخصيص متغير هيكل بيانات لمتغير أخر من نفس النوع. يترتب على ذلك نسخ جميع عناصر الخاصة بأحد المتغيرين إلى العناصر الخاصة بالمتغير الأخر، كما يتضح من الكود التالى:

```
Dim YourChart As AccChart
YourChart = MyChart
```

العلاقة بين هياكل البيانات والمصفوفات

يمكـن أن يحـتوى هيكل البيانات على مصفوفة من بين العناصر التى يتكون منها ، كما فى المثال التالى :

```
Private Structure SystemInfo
  Private CPU As String
  Private Memory As Long
  Private DiskDrives() As String
  Private PurchaseDate As Date
End Structure
```

ويمكننا الوصول إلى قيم المصفوفة التى فى داخل هيكل بيانات بنفس الطريقة المستخدمة للوصول إلى خاصية فى أحد الكائنات ، كما يتضح من الكود التالى :

```
Dim MySystem As SystemInfo
ReDim MySystem.DiskDrives(3)
MySystem.DiskDrives(0) = "1.44 MB"
```

ويمكن أيضا الإعلان عن مصفوفة من هياكل البيانات ، كما يتضح من الكود التالى :

```
Dim AllSystems(100) As SystemInfo
```

ويمكن الوصول إلى بيانات هيكل البيانات من خلال المصفوفة باستخدام الكود التالى :

```
AllSystems(5).CPU = "386SX"
AllSystems(5).DiskDrives(2) = "100M SCSI"
```

العلاقة بين هياكل البيانات وبين الكائنات

يمكـن أن يحـتوى هـيكل البـيانات عـلى كائـنات بـين أعضـائه ، كما يتضح من الكود التالى :

```
Private Structure AccountPack
  Private frmInput as Form
  Private dbPayRollAccount as Database
End Structure
```

العلاقة بين هياكل البيانات وبين الإجراءات

يمكن تمرير متغير هيكل بيانات إلى أحد الإجراءات ، كما يتضح من الكود التالى :

```
Sub FillSystem(ByRef SomeSystem As SystemInfo)
  SomeSystem.CPU = lstCPU.Text
  SomeSystem.Memory = lstCPU.Amount
  SomeSystem.PurchaseDate = Now
End Sub
```

فى المثال السابق، تم تمرير مرجع هيكل البيانات ، مما يمكن الإجراء من تعديل قيم

عناصر هيكل البيانات فى كود الاستدعاء. ويمكن تمرير متغير الهيكل بالقيمة لمنع الإجراء من تغيير قيم عناصر الهيكل فى كود الاستدعاء. ويمكن أيضا إعادة هيكل بيانات من إجراء Function، كما يتضح من الكود التالى:

```
Dim AllSystems(100) As SystemInfo
Function FindByDate(ByVal SearchDate As Date) As SystemInfo
  Dim I As Integer
  For I = 1 To 100
    If AllSystems(I).PurchaseDate = SearchDate Then Return AllSystems(I)
  Next I
End Function
```

العلاقة بين الهياكل وبين التصنيفات

هناك أوجه تشابه واختلاف بين هياكل البيانات والتصنيفات، مع أن Visual Basic .NET يوحد تقريبا القواعد اللغوية المستخدمة مع كليهما. يماثل هيكل البيانات التصنيف فى أن كلاهما يحتوى على أعضاء، إجراءات بناء، وسائل، خصائص، حقول، ثوابت، تعدادات، وأحداث. ويمكن لكليهما تنفيذ الواجهات البينية (Interfaces). وأهم أوجه الاختلافات هى:

- كل أعضاء هيكل البيانات عامة (Public) فى الأصل، بينما تكون المتغيرات والثوابت فى التصنيف خاصة (Private) و باقى الأعضاء عامة (Public) فى الأصل.
- لايمكن إعلان أعضاء هيكل البيانات باستخدام كلمة Protected.
- لا تستطيع إجراءات هيكل البيانات معالج الأحداث.
- الإعلانات عن متغيرات هياكل البيانات لا تستطيع تحديد قيم ابتدائية، استخدام كلمة New، أو تحديد أحجام ابتدائية للمصفوفات.
- لايمكن وراثة هياكل البيانات.

تحويل أنواع البيانات

يعنى تحويل أنواع البيانات (Type Conversion) تحويل قيمة من أحد أنواع البيانات إلى نوع بيانات أخر. مثل هذه التحويلات ممكنة لأن هناك الكثير من القيم التى يمكن تخزينها باستخدام أكثر من نوع من أنواع البيانات. على سبيل المثال، القيمة ١٢٣٤٥٦٧٨٩١ يمكن تخزينها فى صورة Integer أو فى صورة long. ويرتبط بعملية تحويل

الأنواع مصطلحان مهمان: تحويل التوسيع (Widening Conversion)، الذى يعنى تحول قيمة إلى نوع يحتوى على عدد أكبر من وحدات بت (Bits)، مثل تحويل نوع Integer إلى نوع Long. والنوع الثاني من التحويل هو تحويل التضييق (Narrowing Conversion)، الذى يعنى تحويل قيمة إلى نوع يحتوى على عدد أقل من وحدات بت، مثل تحويل قيمة من نوع Long إلى نوع Integer.

تحويل التوسيع

يقوم تحويل التوسيع بتغيير قيمة إلى نوع بيانات يمكن أن يتوافق مع أى قيمة ممكنة من البيانات الأصلية. الأمثلة التالية، تعرض بعض تحويلات التوسيع القياسية:

- التحويلات من أى نوع مشتق إلى الأنواع الأصلية التى تم منها الاشتقاق.
- التحويلات من Byte إلى Short إلى Integer إلى Long إلى Decimal إلى Single إلى Double.
- التحويل من قيمة Nothing إلى أى قيمة أخرى.
- التحويل من صفر إلى أى نوع عددى.
- التحويلات من نوع Char إلى نوع String.
- التحويل من أى نوع إلى Object.

تحويل التضييق

يقوم تحويل التضييق (Narrowing Conversions) بتحويل قيمة إلى نوع بيانات قد لا يكون قادرا على الاحتفاظ ببعض قيم البيانات الأصلية، وهى تحويلات ترتبط بفقدان المعلومات أحيانا. من أمثلة هذه التحويلات:

- التحويلات فى الاتجاهات المضادة لتحويلات التوسيع.
- التحويلات بين الأنواع العددية والأنواع المنطقية فى كلا الاتجاهين
- التحويلات بين مصفوفة من نوع Char وبين نوع السلسلة فى كلا الاتجاهين.
- التحويلات بين نوع السلسلة وبين أى نوع عددى ، نوع منطقى، ونوع التاريخ فى كلا الاتجاهين.

• التحويل من نوع بيانات إلى نوع بيانات مشتق منة.

كلمات تحويل الأنواع

يؤثر استخدام عبارة Option Strict تأثيرا كبيرا على تحويل التضييق (Narrowing Conversions) بين الأنواع. عندما يكون Option Strict فعال، فإن برنامج الترجمة (Compiler) لن يسمح بتحويل التضييق الضمني، ولكنة يسمح بتحويل التضييق الصريح. ويتم تنفيذ تحويل التضييق الصريح باستخدام الكلمات المرشدة (Keywords) الخاصة بتحويل الأنواع. ويستخدم مصطلح Casting أيضا، لتعريف تحويل تضييق الأنواع. من أمثلة الكلمات المستخدمة فى إنجاز عمليات تحويل الأنواع الصريحة كلمة Cint، التى تقوم بتحويل القيمة إلى نوع Integer، كما يتضح من الكود التالى:

```
Dim L As Long = 1254245454
Dim  I As Integer = Cint(L)
```

يقوم الكود السابق بتحويل قيمة من نوع Long إلى قيمة من نوع Integer بغض النظر عن استخدام عبارة Option Strict. ويبين الجدول رقم (١١) كلمات تحويل الأنواع المختلفة التى يمكن استخدامها.

كلمة التحويل	النوع المحول إلية	النوع المسموح بتحويلة
CBool	Boolean	الأنواع العددية ، السلاسل، الكائن
CByte	Byte	الأنواع العددية ، المنطقية ، السلاسل، الكائن
CChar	Char	السلسلة ، الكائن
CDate	Date	السلسلة ، الكائن
CDbl	Double	الأنواع العددية ، المنطقية ، السلسلة ، الكائن
CDec	Decimal	الأنواع العددية ، المنطقية ، السلسلة ، الكائن
CInt	Integer	الأنواع العددية ، المنطقية ، السلسلة ، الكائن
CLng	Long	الأنواع العددية ، المنطقية ، السلسلة ، الكائن

كلمة التحويل	النوع المحول إليه	النوع المسموح بتحويلة
CObj	Object	أى نوع
CShort	Short	الأنواع العددية ، المنطقية ، السلسلة ، الكائن
CSng	Single	الأنواع العددية ، المنطقية ، السلسلة ، الكائن
CStr	String	الأنواع العددية ، المنطقية ، السلسلة ، الكائن
CType	النوع الذى يلى الفاصلة	عند التحويل إلى أنواع بيانات ابتدائية ، يكون نفس النوع المسموح به مع كلمة التحويل المقابلة

جدول ١١

ويأخذ التركيب اللغوي لاستخدام كلمات التحويل، فيما عدا CType، الصيغة التالية:

[Property of Variable] = [Type conversion keyword] (value being converted)

وتستخدم كلمة CType صيغة لغوية مخالفة، تحتوى على معاملين بدلا من معامل واحد. يمثل المعامل الأول، التعبير الذى يجب تحويلة، والمعامل الثاني يمثل نوع البيانات المقصود أو تصنيف الكائن (Object Class). ويوضح الكود التالى كيفية استخدام كلمة CType:

```
Dim MyInt As Integer = CType (MyOtherInt, Integer)
```

ولا تستطيع كلمات تحويل الأنواع منع أخطاء تجاوز النوع أو تجنب التقريب. على سبيل المثال، يترتب على تنفيذ الكود التالى حدوث خطأ تجاوز حدود القيمة التى يمكن أن يحتويها نوع Integer (Overflow error):

```
Dim L As Long = 2147483648
Dim I As Integer = CInt (L)
```

على هذا الأساس، لا تقوم كلمات تحويل الأنواع بالتحقق من أن قيمة المصدر صالحة لنوع بيانات المقصد. وسوف يحدث خطأ إذا لم تكن القيمة التى يتم تخصيصها صالحة، مثل أن تكون القيمة التى يجرى تحويلها أضخم من طاقة نوع المقصد على استيعابها. كما

هو موضح فى المثال السابق. وتنحصر فائدة عبارة Option Strict فى هذه الحالة فى توجيه إنتباة المبرمج إلى أنة يقوم باستخدام تحويل التضييق (Narrowing Conversion). وبدون استخدام عبارة Option Strict، يمكن أن نقوم باستخدام تحويل التضييق بدون أن نشعر بذلك.

العوامل (Operators)

تستخدم العوامل (Operators) فى تنفيذ العمليات المختلفة على القيم. وتستخدم فى تكوين التعبيرات (Expressions) عن طريق الربط بين المتغيرات والثوابت والتعبيرات الأخرى. وتنقسم العوامل إلى الأنواع التالية:

- عوامل التخصيص (Assignment Operators)
- العوامل الحسابية (Arithmetic Operators)
- عوامل المقارنة (Comparison Operators)
- عوامل التسلسل (Concatenation Operators)
- العوامل المنطقية والتعامل مع وحدات بت (Logical/Bitwise Operators)
- عوامل متنوعة (Miscellaneous Operators)

عوامل التخصيص

تستخدم علامة (=) للقيام بوظيفتين فى Visual Basic .NET. الوظيفة الأولى هى القيام بعملية التخصيص، والوظيفة الثانية هى المقارنة. يستخدم عامل التخصيص لتحديد قيمة لأحد المتغيرات عند الإعلان أو بعد الإعلان عنة، كما يتضح من الكود التالى:

```
Dim intVar As  Integer = 1
```

فى هذه الحالة، تتم عملية التخصيص على نفس المستوى الذى يتم فيه الإعلان عن المتغير، سواء تم الإعلان عن المتغير على مستوى وحدة الكود أو فى داخل أحد الإجراءات. وعلى العكس من ذلك، عند تخصيص قيمة لمتغير بعد الإعلان عنة، فإن عملية التخصيص يجب أن تحدث فقط داخل أحد الإجراءات، بغض النظر عن مستوى الإعلان عن المتغير.

الجانب الأيسر فى عبارة التخصيص

الغرض من عبارة التخصيص هو تغيير قيمة المتغير أو الخاصية التى على يسار عبارة

التخصيص. وعلى هذا الأساس، يجب أن يكون المتغير أو الخاصية التى على يسار عبارة التخصيص قابلة لتغيير قيمتها. والمتغير بطبيعته يقبل هذا التغيير، على عكس الثابت. ولهذا لايمكن أن يوضع ثابت على يسار عبارة التخصيص. وبالمثل لايمكن وضع قيم حرفية، مثل ٢١٥ أو "Hello" على يسار عبارة التخصيص. وعلى خلاف المتغيرات، هناك بعض الخصائص التى لا تقبل تغيير قيمتها. على سبيل المثال، معظم أدوات التحكم على سطح النموذج يمكن اختيارها بواسطة الماوس. غير أن هناك بعض أدوات التحكم التى لايمكن اختيارها، مثل متحكم Label. وعندما تكون الخاصية قابلة لتغيير قيمتها، فإن عبارة تخصيص القيمة لهذه الخاصية تماثل العبارة المستخدمة مع المتغير، فيما عدا أن الخاصية يجب تأهيلها باسم الكائن الذى تتبعة.

الجانب الأيمن فى عبارة التخصيص

يمكن أن يحتوى الجانب الأيمن فى عبارة التخصيص على أى شئ له قيمة، مثل الخاصية، المتغير، الثابت، قيمة حرفية، أو تعبير. والشرط الوحيد فى هذا السياق هو أن تكون الخاصية قابلة للقراءة فى وقت تشغيل التطبيق. ويجب أن تكون القيمة التى على يمين عبارة التخصيص من نوع بيانات يمكن تخزينه فى المتغير أو الخاصية التى على يسار عبارة التخصيص. ولن تبرز هذه المشكلة إذا كان نوع البيانات على يمين عبارة التخصيص يماثل نوع البيانات على يسار تلك العبارة. وفى حالة اختلاف أنواع البيانات بين الجانبين، سوف يتوقف نجاح تنفيذ العبارة على عاملين. العامل الأول هو ما إذا كان نوع البيانات الذى على يمين عبارة التخصيص يقبل التحويل إلى نوع البيانات الذى على الجانب الأيسر من علامة التخصيص. والعامل الثانى يتعلق بما إذا كان Visual Basic .NET سوف يسمح بعملية التحويل بين النوعين.

العوامل الحسابية

تستخدم العوامل الحسابية (Arithmetic Operators) فى الكثير من العمليات الحسابية المعتادة، التى تشمل القيم العددية الممثلة بالقيم المجردة، المتغيرات، التعبيرات، الثوابت، استدعاءات الدوال والخصائص. وتحتاج العوامل الحسابية إلى معاملين (Operands)، أحدهما على يسار العامل (Operator)، والأخر على اليمين. الجدول رقم (١٢) يعرض العلامات الحسابية والوظائف التى تقوم بها.

العامل	العملية
^	الرفع إلى قوة (الأس)
-	النفى (تغيير إشارة الأرقام)
*, /	القسمة والضرب
\	قسمة الأرقام الصحيحة
Mod ()	باقى القسمة
+, -	الطرح والجمع

جدول ١٢

عوامل المقارنة

تقوم عوامل المقارنة (Comparison Operators) بمقارنة تعبيرين وإعادة قيمة منطقية تمثل نتيجة المقارنة. ويمكن تقسيم هذه العوامل إلى ثلاثة مجموعات: عوامل مقارنة القيم العددية، عوامل مقارنة السلاسل، وعوامل مقارنة الكائنات.

مقارنة القيم العددية

هناك ستة عوامل تستخدم فى مقارنة القيم العددية. يعرض الجدول رقم (١٣) هذه العوامل، كما يعرض مثالا على الشرط الذى يقوم العامل باختباره :

العامل	الشرط	المثال
يساوى =	القيمة الأولى تساوى القيمة الثانية	٢٥=٢١ غير حقيقى ٢٥=٢٥ حقيقى ٢٥=٢٦ غير حقيقى
لا يساوى <>	القيمة الأولى لا تساوى القيمة الثانية	٢٥<>٢١ حقيقى ٢٥<>٢٥ غير حقيقى ٢٥<>٢٦ حقيقى
أقل من <	القيمة الأولى أقل من القيمة الثانية	٢٥<٢١ غير حقيقى ٢٥<٢٥ غير حقيقى

العامل	الشرط	المثال
		٢٥<٢٦ حقيقى
أكبر من >	القيمة الأولى أكبر من القيمة الثانية	٢٥>٢١ حقيقى ٢٥>٢٥ غير حقيقى ٢٥>٢٦ غير حقيقى
أقل من أو يساوى <=	القيمة الأولى أقل من أو تساوى القيمة الثانية	٢٥<=٢١ غير حقيقى ٢٥<=٢٥ حقيقى ٢٥<=٢٦ حقيقى
أكبر من أو يساوى >=	القيمة الأولى أكبر من أو تساوى القيمة الثانية	٢٥>=٢١ حقيقى ٢٥>=٢٥ حقيقى ٢٥>=٢٦ غير حقيقى

جدول ١٣

مقارنة سلاسل الرموز

يمكن مقارنة سلاسل الرموز بإحدى طريقتين: استخدام العامل Like، واستخدام عوامل المقارنة العددية. يسمح لنا العامل Like بتكوين نموذج، ثم المقارنة مع ذلك النموذج. وعندما يتساوى النموذج مع سلسلة المقارنة، يكون الناتج True. وإلا، فإن النتيجة تكون False. وتسمح لنا عوامل المقارنة العددية بمقارنة قيم سلسلة على أساس ترتيب هذه القيم، كما فى المثال التالى :

"85" < "9"

نتيجة المقارنة السابقة تكون True لأن الرمز الأول فى السلسلة التى على اليسار، وهو "8" ، يأتى فى الترتيب قبل الرمز الأول من السلسلة التى على اليمين. وإذا تساوى الرمز الأول فى كلا السلسلتين، تستمر المقارنة بين الرمز الثانى فى كلا السلسلتين، وهكذا. وعند المقارنة بين سلسلتين تمثل إحداهما بادئة للسلسلة الأخرى، تأتى السلسلة الأطول فى ترتيب الفرز بعد السلسلة الأخرى. على هذا الأساس، يكون التعبير التالى حقيقيا :

"kkk" > "kk"

ويعتمد ترتيب الفرز على المقارنة الثنائية (Binary Comparison) أو على أساس

المقارنة النصية (Textual Comparison) بناءا على القيمة المذكورة فى عبارة Option Compare السابق الحديث عنها.

مقارنة الكائنات

يمكن استخدام عامل Is لتقرير ما إذا كان هناك متغيرين يستخدمان مرجع إلى نفس كائن التصنيف. كما يتضح من المثال التالى:

```
Dim x As MyClass
Dim y As New MyClass()
x = y
If x Is y Then
```

فى هذا المثال، المتغيران x و y يستخدمان مرجع إلى نفس كائن تصنيف MyClass. ولهذا تكون نتيجة المقارنة فى العبارة الأخيرة من الكود حقيقية. وعكس هذا المثال يوضحه المثال التالى:

```
Dim x As New MyClass()
Dim y As New MyClass()
If x Is y then
```

فى هذا المثال، العبارة الأخيرة فى الكود غير حقيقية لأن المتغير x والمتغير y يستخدمان مراجع إلى أمثلة مختلفة من التصنيف.

ويمكن اختبار ما إذا كان أحد الكائنات من نوع معين باستخدام عامل TypeOf...Is، كما سبق توضيحه فى الفصل السابق. ويأخذ التعبير الذى يحتوى على هذا العامل الصيغة التالية:

```
TypeOf <object expression> Is <TypeName>
```

وعندما يكون TypeName من نوع واجهة بينية (Interface)، يعيد العامل TypeOf...Is القيمة True إذا كان الكائن ينفذ هذا النوع من الواجهات البينية. وعندما يكون TypeName أحد التصنيفات، يعيد إلينا هذا العامل القيمة True إذا كان الكائن مثل من هذا التصنيف أو مثل من تصنيف مشتق من التصنيف المحدد فى العبارة. لتوضيح ذلك، نعرض المثال التالى:

```
Dim x As Button
x = New Button()
If TypeOf x Is Control Then
```

العبارة الأخيرة فى الكود السابق حقيقية لأن المتغير x يشير إلى مثل من تصنيف Button المشتق من تصنيف Control.

عوامل التسلسل (Concatenation Operators)

تقوم عوامل التسلسل (Concatenation Operators) بربط عدد من السلاسل معا لتكوين سلسلة واحدة. وهناك اثنان من عوامل الربط المستخدمة فى هذه الوظيفة، عامل + وعامل &. يبين الكود التالى استخدام هذين العاملين فى عمليات ربط السلاسل:

```
Dim x As String
x = "Con" & "caten" & "ation"
x = "Con" + "caten" + "ation"
```

بعد تنفيذ المثال السابق، سوف تحتوى x على سلسلة "Concatenation". ويمكن لتلك العوامل أيضا ربط المتغيرات من نوع String، كما فى المثال التالى:

```
Dim x As String = "act"
Dim y As String = "def"
Dim z As String
z = x & y
z = x + y
```

فى المثال السابق، سوف تحتوى z على سلسلة "abcdef".

ويمكن الربط بين السلاسل الحرفية (String Literals) وبين المتغيرات أو التعبيرات، كما يتضح من المثال التالى:

```
Debug.Write("2 Squared is  " & 2^2)
```

العوامل المنطقية (Logical Operators)

تقوم العوامل المنطقية (Logical Operators) بمقارنة التعبيرات المنطقية (Boolean Expressions) وإعادة نتيجة منطقية لهذه المقارنة. وتأخذ عوامل And، Or، AndAlso، OrElse، و Xor، اثنين من المعاملات، بينما يأخذ عامل Not معاملا واحدا. ويقوم عامل Not بتنفيذ عملية نفى منطقى لأحد التعبيرات المنطقية. بمعنى أخر، ينتج عنة عكس قيمة التعبير الذى يقوم بتقييمه. فإذا كان التعبير قيمته True، فإن عامل Not يؤدى إلى جعلة False. وإذا كانت قيمة التعبير False، يقوم Not بجعل القيمة True. المثال التالى يوضح ذلك:

```
Dim x As Boolean
```

```
x = Not 23 > 12
x = Not 23 > 67
```

التعبير الثانى فى الكود السابق جعل ٢٣ ليست أكبر من ١٢ وبالتالى أصبح التعبير غير حقيقى. والتعبير الأخير جعل ٢٣ ليست أكبر من ٦٧ وبالتالى أصبح التعبير حقيقيا.

ويقوم عامل And بالضم المنطقى لاثنين من التعبيرات المنطقية. يعنى ذلك، أنة إذا كان كل من التعبيرين حقيقى (True)، فإن نتيجة And تكون True أيضا. وإذا كان أى من التعبيرين أو كلاهما غير حقيقى (False)، فإن نتيجة And تكون False أيضا. وعند استخدام عامل Or للربط بين تعبيرين منطقيين، إذا كان أى من التعبيرين حقيقيا (True)، تكون نتيجة Or حقيقية (True). وإذا كان كلا التعبيرين غير حقيقى، تكون نتيجة Or غير حقيقية. وتقوم Xor بعملية تحقيق الانفراد المنطقى على تعبيرين. إذا كان أى من التعبيرين حقيقيا، فإن نتيجة Xor تكون حقيقية. وإذا كان كلا التعبيرين حقيقيا أو غير حقيقى، فإن نتيجة Xor تكون غير حقيقية.

ويماثل العامل AndAlso العامل And، فى أنة يقوم بعملية وصل منطقية لتعبيرين منطقيين. الفرق الجوهرى بين العاملين هو أن AndAlso تقوم بتقييم التعبير الأول، فإذا كان غير حقيقى لا يتم تقييم التعبير الثانى وتكون القيمة العائدة من AndAlso غير حقيقية. وبالمثل، إذا كان تقييم التعبير الأول بالنسبة لعامل OrElse حقيقيا، لا تقوم OrElse بتقييم التعبير الثانى وتكون نتيجة OrElse حقيقية. فيما يلى بعض الأمثلة التى توضح الفرق بينهما:

التعبير التالى يؤدى إلى استدعاء دالة ()MyFunction لأن استدعاء الدالة لا يتوقف على تقييم التعبير الأول

```
12 > 45 And MyFunction (4)
```

التعبير التالى يؤدى إلى عدم استدعاء دالة ()MyFunction لأن استدعاء الدالة يرتبط بتقييم التعبير الأول

```
12 > 45 AndAlso MyFunction (4)
```

التعبير التالى يؤدى أيضا إلى استدعاء دالة ()MyFunction لعدم توقف العبارة الثانية على الأولى

```
45 > 12 Or MyFunction(4)
```

التعبير التالى يؤدى إلى عدم استدعاء دالة MyFunction() لأن التعبير الثانى لا يتم تقييمه إذا كان التعبير الأول صحيحا.

```
45 > 12 OrElse MyFunction (4)
```

وتستخدم العوامل المنطقية أيضا فى تنفيذ العمليات التى تجرى على أرقام النظام الثنائي (Bits). وتقوم هذه العمليات بتقييم قيمتين على أساس النظام الثنائى (Binary System)، مقارنة الأرقام الثنائية فى الأماكن المتقابلة، ثم تخصيص قيم بناءا على عملية المقارنة، كما فى الإيضاح التالى الذى يستخدم عامل And:

```
Dim x As Integer
x = 3 And 5
```

فى المثال السابق، قيمة x تساوى واحد. ويرجع السبب فى ذلك إلى ما يلى:

- فى البداية، يتم تحويل القيم إلى الشكل الثنائى، الموضح فيما يلى :

```
3 = 011
5 = 101
```

- الخطوة التالية، تتم المقارنة بين القيمة فى صورتها الثنائية، وتتم المقارنة على أساس موقع ثنائى فى كل مرة. إذا كان الرقمان فى الموقعين يساوى كل منهما واحد، يتم وضع واحد فى الموقع المقابل بالنتيجة. وإذا كان كلاهما صفر، يتم وضع صفر فى الموقع المقابل بالنتيجة، وإذا كان الرقمان مختلفان، يتم وضع صفر فى الموقع المقابل بالنتيجة.

```
011     3 in binary form
101     5 in binary form
001    The result, in binary form
```

- يتم بعد ذلك، تحويل القيمة الثنائية 001 إلى قيمة عشرية تساوى 1.

ويماثل استخدام Or فى العمليات الثنائية، استخدام And، فيما عدا أن القيمة التى يتم تخصيصها للموقع المقابل فى النتيجة يساوى واحدا، إذا كانت قيمة أى من الرقمين الثنائيين اللذين يتم مقارنتهما تساوى واحدا. وعند استخدام Xor ، يتم تخصيص واحد للنتيجة فقط عندما يكون أحد الرقمين يساوى واحدا وليس كلاهما. وتقوم Not بعكس جميع القيم الثنائية فى المعامل الوحيد معها، بما فى ذلك الرقم الدال على الإشارة، وتضع القيمة بعد ذلك فى النتيجة. ولذلك تكون نتيجة Not مع الأرقام السلبية موجبة أو صفر ومع الأرقام الموجبة سلبية. ويجب ملاحظة أن هذه الأنواع من العمليات تجرى فقط على الأعداد

الصحيحة.

العوامل المركبة من عامل التخصيص والعوامل الاخرى

يمكن تركيب بعض العوامل وعامل التخصيص، معا، مما يؤدى إلى اختصار عدد العبارات التى نقوم بكتابتها لتحقيق مهمة محددة. الجدول رقم (١٤)، يعرض العوامل الحسابية والعوامل المنطقية المركبة مع عامل التخصيص.

العامل	الاستخدام	العبارات
+=	الزيادة بمقدار واحد	intVar += 1
-=	التنزيل بمقدار واحد	intVar -= 1
*=	المضاعفة	intVar *= 2
/=	التنصيف	intVar /= 2
\=	التنصيف	intVar \= 2
^=	التربيع	intVar ^= 2
&=	الإلحاق	strName &= strAppendName
<=	المقارنة	intVar1 <= intVar2
>=	المقارنة	intVar1 >= intVar2

جدول ١٤

من الملاحظ فى الجدول رقم (١٤) عدم وجود تركيب مختصر يمثل عامل باقى القسمة (Mode). يرجع السبب فى ذلك إلى أن باقى قسمة متغير على نفسة يساوى صفر.

ترتيب اولويات العوامل

عند وقوع عدد من العمليات فى أحد التعبيرات، يتم تقييم كل جزء على أساس ترتيب مقرر سلفا يطلق علية أولوية العامل (Operator Precedence). وعندما يحتوى التعبير على عوامل من مجموعات مختلفة، يجرى تقييمها أولا على أساس الترتيب التالى :

١. العوامل الحسابية وعوامل وصل السلاسل (Arithmetic and Concatenation Operators)

٢. عوامل المقارنة (Comparison Operators)

٣. العوامل المنطقية (Logical Operators)

وتمتلك كل عوامل المقارنة نفس الدرجة من الأولوية، ويتم تقييمها بترتيب ظهورها من اليسار إلى اليمين. وتختلف أولويات العوامل الحسابية، عوامل وصل السلاسل، والعوامل المنطقية طبقا للجدول رقم (١٥):

ترتيب الأولوية	العامل	إيضاح
١	^	الرفع إلى قوة معينة (الأس)
٢	-	تغيير إشارة القيم (النفى)
٣	*, /	الضرب وقسمة قيم العلامة العشرية المتحركة
٤	\	قسمة الأرقام الصحيحة
٥	Mod ()	عمليات باقى القسمة
٦	+, -	الإضافة والطرح، وربط السلاسل (+)
٧	&	ربط السلاسل لتكوين سلسلة واحدة
٨	Not	نفى التعبيرات بتغيير إشارتها
٩	And, AndAlso	الربط بين التعبيرات المنطقية
١٠	Or, OrElse, Xor	الفصل بين التعبيرات بين التعبيرات المنطقية

جدول ١٥

وبصفة عامة يتم التقييم من اليسار إلى اليمين عند حدوث عمليات لها نفس الأولوية، مثل عمليات الضرب والقسمة. كما يمكن التحكم فى هذه الأولويات عن طريق استخدام الأقواس. لأن العوامل التى داخل الأقواس لها أولوية فى التنفيذ على العوامل التى خارجها. وفى داخل الأقواس يجرى استخدام نفس القواعد التى تحكم أولويات تنفيذ العوامل المختلفة. ومع أن عامل (&) ليس من العوامل الحسابية، إلا أنة يأتى بعدها مباشرة فى الأولوية ويسبق عوامل المقارنة والعوامل المنطقية.

التعبيرات

يمكن تعريف التعبيرات (Expressions) بأنها سلاسل من القيم (Values) التى تفصل العوامل (Operators) بينها، ويؤدى التعبير إلى توليد قيمة جديدة. من أمثلة القيم التى تكون التعبيرات، تركيبات الحروف، المتغيرات، التعبيرات الأخرى، واستدعاء الدوال. وتقوم العوامل بإجراء العمليات على عناصر القيم، مثل العمليات الحسابية، المقارنات، وغيرها من العمليات. وتمثل النتيجة النهائية للتعبير القيمة، التى يمكن أن تكون عددية، منطقية، أو من قيم السلاسل.

تعبيرات العمليات الحسابية

يمكن إجراء العمليات الحسابية على القيم العددية من خلال التعبيرات العددية. ويمكن تعريف التعبير العددى بأنة تعبير يحتوى على تركيبات حروف ومتغيرات تمثل قيم عددية، وعلى عوامل تعمل على هذه القيم. المثال التالى،يعرض أحد هذه التعبيرات:

```
5 * (12 + x)
```

عندما نفترض أن قيمة x تساوى ٣، فإن قيمة التعبير السابق تساوى ٧٥. يرجع ذلك إلى أن Visual Basic يقوم بتقييم التعبير الذى بين الأقواس أولا ثم تنفيذ عملية الضرب. ويتم تقييم التعبيرات العددية التى تحتوى على أكثر من عامل على أساس القواعد التى تحكم أولويات العوامل. لكى يمكننا تجاهل هذه القواعد، نضع التعبيرات بين أقواس، كما فى المثال السابق. يمكن بعد ذلك استخدام عبارة تخصيص لوضع قيمة التعبير السابق فى متغير أخر، كما فى المثال التالى:

```
Dim x As Integer = 3
Dim y As Integer
y = 5 * (12 + x)
```

فى هذا المثال، تصبح قيمة y مساوية ٧٥ نتيجة تخصيص قيمة التعبير الذى على يمين عامل التخصيص للمتغير الذى على يسار هذا العامل.

تعبيرات عمليات المقارنة

تستخدم عوامل المقارنة فى بناء التعبيرات التى تحتوى على مقارنات بين متغيرات من أنواع القيم العددية. وينتج عن هذه التعبيرات قيمة منطقية (Boolean Value) توضح ما إذا كانت المقارنة حقيقية (True) أو غير حقيقية (False). ويمكن أن يقوم تعبير المقارنة بمقارنة تعبيرات جزئية داخلية، مما يجعل هذا النوع من التعبيرات أكثر تعقيدا. من أمثلة هذه

التعبيرات ما يلى:

```
x / 45 * (y +17) >= System.Math.Sqrt (z) / (p – (x * 16))
```

يشتمل التعبير السابق على قيم حرفية، متغيرات، كما يشتمل على استدعاء لأحد الدوال. ويتم تقييم التعبيرات التى يحتوى عليها على كلا جانبى عامل المقارنة ثم المقارنة بين النتائج باستخدام عامل المقارنة =>. فإذا كانت قيمة التعبير الذى على الجانب الأيسر أكبر من أو يساوى قيمة التعبير الذى على الجانب الأيمن، فإن قيمة التعبير بالكامل تساوى True؛ وإلا، فإن قيمة هذا التعبير تصبح False.

ويمكن استخدام تعبيرات المقارنة فى موقع نحتاج فيه إلى استخدام القيم المنطقية (Boolean Values)، كما هو الحال مع عبارات If، While، Loop، ElseIf أو عند تخصيص أو تمرير قيمة إلى متغير منطقى. فى المثال التالى، يتم تخصيص القيمة الناتجة من عملية المقارنة على يمين عامل التخصيص للمتغير الذى على يسار العامل:

```
Dim N As Boolean
N = 50 < 30
```

تعبيرات العمليات المنطقية

التعبير المنطقى هو التعبير الذى يترتب على تقييمه الحصول على قيمة منطقية (Boolean Value)، مثل القيمة True أو القيمة False. ويمكن أن تأخذ التعبيرات المنطقية أشكالا عديدة. أبسط هذه الأشكال يشتمل على المقارنة المباشرة بين متغير منطقى (Boolean Variable) وبين قيمة حرفية منطقية (Boolean Literal)، كما يتضح من المثال التالى:

```
If x = True Then
    y = False
End If
```

فى المثال السابق، تم استخدام عامل = فى عبارة If بصفته عامل مقارنة بين القيمة True وبين المتغير x. وفى السطر الثانى من الكود، يستخدم عامل = بصفته عامل تخصيص القيمة المنطقية False للمتغير y .

عوامل المقارنة، مثل =، >، <، <>، =>، و =<، تقوم بتكوين التعبيرات المنطقية عن طريق المقارنة بين التعبير الذى الجانب الأيسر مع التعبير الذى على الجانب الأيمن من العامل المنطقى وتقييم النتيجة True أو False. ويمكن تكوين تعبيرات منطقية أكثر تعقيدا

باستخدام تعبيرات المقارنة والعوامل المنطقية التى تفصل بينها. المثال التالى يوضح استخدام عوامل المقارنة مع عامل منطقى لتكوين تعبير منطقى أكثر تعقيدا:

```
x > y And x < 1000
```

فى المثال السابق، تعتمد القيمة النهائية للتعبير الأساسى على قيم التعبيرات الموجودة على كل جانب من جوانب العامل المنطقى And. إذا كان كل من التعبيرين حقيقيا، تكون النتيجة النهائية حقيقية. وإذا كانت قيمة إحدى التعبيرين غير حقيقية، فإن القيمة النهائية للتعبير الأصلى تكون غير حقيقية.

ويمكن استخدام عوامل الجولات القصيرة (Short-Circuiting Operators) أيضا فى تكوين التعبيرات المنطقية. تشمل هذه العوامل التى سبق إيضاحها، عامل AndAlso وعامل OrElse. من أمثلة التعبيرات المنطقية التى تشتمل على AndAlso، الكود التالى :

```
If 45 < 12 AndAlso MyFunction(3) = 81 Then
End If
```

فى هذا المثال، يقوم العامل بتقييم الجانب الأيسر، وبالنظر إلى أن قيمة الجانب الأيسر غير حقيقية (False)، فإن كامل التعبير يصبح غير حقيقى. يترتب على تنفيذ الكود السابق، عدم استدعاء الدالة ()MyFunction لأن قيمة التعبير الذى على يسار العامل المنطقى تساوى False. وبالمثل، عندما تكون قيمة التعبير الذى على الجانب الأيسر من العامل المنطقى OrElse حقيقية (True)، يتم إهمال تقييم التعبير الذى على الجانب الأيمن من التعبير لأن التعبير قد أصبح حقيقيا عندما وجدنا أن قيمة التعبير الذى على الجانب الأيسر تساوى True. وعلى خلاف عوامل الجولات القصيرة، يقوم كل من عامل And وعامل Or بتقييم كل من التعبيرين الموجودين على يسار ويمين العامل قبل تقرير قيمة التعبير الأساسى. على سبيل المثال، يجرى استدعاء الدالة التى على يمين العامل المنطقى And فى التعبير التالى، مع أن قيمة التعبير الذى على يسار العامل يساوى False.

```
If 45 < 12 And MyFunction(3) = 81 Then

End If
```

ويمكن استخدام الأقواس للتحكم فى عمليات تقييم التعبيرات المنطقية. حيث يتم تقييم التعبيرات الموجودة داخل أقواس أولا. وعندما يكون هناك مستويات متعددة من التعبيرات

المتداخلة، يجرى تقييم التعبير الأكثر تداخلا أولا. وداخل الأقواس تطبق القواعد التى تحكم أولويات تقييم التعبيرات.

التعليمات

يمكن أن تحتوى التعليمات (Statements) فى Visual Basic على كلمات مرشدة (Keywords)، عوامل (Operators)، ثوابت (Constants)، وتعبيرات (Expressions). ويمكن تقسيم عبارات التعليمات إلى المجموعات التالية:

- تعليمات الإعلانات (Declaration Statements)، التى تقوم بتعريف متغير، ثابت، أو إجراء ويمكن أيضا أن تحدد نوع بيانات.
- تعليمات التخصيص (Assignment Statements)، التى تقوم بتخصيص القيم للمتغيرات والخصائص.
- التعليمات التنفيذية (Executable Statements)، التى تبدأ عمليات التنفيذ. يمكن لهذه التعليمات تنفيذ وسيلة أو دالة، ويمكنها أن تنفذ الأفعال فى حلقة متكررة أو أن تتفرع خلال مجمعات من الكود.

ويمكن وضع أكثر من عبارة من عبارات التعليمات على نفس السطر والفصل بينها باستخدام رمز الوقف (:)، كما يتضح من المثال التالى:

```
Dim MyString As String = "Hello World": MsgBox(MyString)
```

وفى حالة استمرار عبارة تعليمات على أكثر من سطر، يمكن متابعة العبارة على السطر التالى باستخدام رمز استمرار السطر (_) متبوعة بالضغط على مفتاح الإدخال. فى المثال التالى تستمر عبارة MsgBox على سطرين :

```
Public Sub DemoBox ()
  Dim myVar As String
  myVar = "John"
  MsgBox("Hello " & myVar & _
    ". How are you?")
End Sub
```

ولتوثيق الكود الذى نقوم بكتابته، يمكننا استخدام الملاحظات داخل الكود. حيث يمكن لهذه الملاحظات أن تقوم بشرح إجراء أو تعليمات معينة أمام أى فرد يقوم بقراءة أو

العمل مع ذلك الكود بعد ذلك. وعند ترجمة البرامج، يقوم Visual Basic بإهمال هذه الملاحظات. وتبدأ الملاحظات بكلمة REM أو بالفاصلة العلوية (`) متبوعة بمسافة، ويمكن إضافتها فى أى مكان بالكود. ولإلحاق ملحوظة بعبارة تعليمات، ندرج فاصلة علوية أو كلمة REM، متبوعة بالملحوظة. وقد نواجه بوجود خط أزرق متموج أسفل الكود بعد الانتهاء من كتابته. يعنى ذلك أن هناك خطأ ما فى ذلك الكود، يجب تصحيحة قبل ترجمة البرنامج. ويمكن الحصول على رسالة توضح هذا الخطأ بوضع مؤشر الماوس لبعض الوقت على الكلمة التى بها الخطأ.

تعليمات الإعلان (Declaration Statements)

نستخدم تعليمات الإعلان (Declaration Statements) لتسمية وتعريف إجراءات (Procedures)، متغيرات (Variables)، وثوابت (Constants). وعند الإعلان عن إجراء، متغير، أو ثابت، فإننا فى نفس الوقت نحدد نطاق الرؤية (Scope) الخاص به، بناءا على مكان وضع الإعلان والكلمات المرشدة التى نستخدمها فى الإعلان عنة. يحتوى المثال التالى على ثلاثة إعلانات:

```
Public Sub ApplyFormat()
  Const limit As Integer = 33
  Dim myJob As Jobs


End Sub
```

فى الكود السابق، تقوم عبارة Public Sub مع عبارة End Sub المرتبطة بها بالإعلان عن إجراء يسمى ApplyFormat. ويتم تنفيذ جميع التعليمات التى تقع بين عبارة Public Sub وبين عبارة End Sub عند تشغيل أو استدعاء هذا الإجراء. وفى داخل الإجراء، تعلن عبارة Const عن الثابت limit، مع تحديد نوع البيانات والقيمة المخصصة له. العبارة الأخيرة، وهى عبارة Dim تعلن عن المتغير myJob ونوع بياناته هو تصنيف Jobs. وتمثل الكلمة المرشدة Dim واحدة من أنواع الكلمات المستخدمة فى الإعلان عن المتغيرات. وكما سبق إيضاحه، هناك كلمات أخرى، مثل Redim، Static، Public، Private، Protected، و Friend.

وتقوم تعليمات الإعلان بمهمة حجز الذاكرة التى يحتاجها البرنامج لتكوين متغير،

ولكنها لا تقوم بتكوين ذلك المتغير. ويمكن التكوين الصريح للمتغير وتخصيص قيمة له فى نفس الوقت باستخدام التركيب اللغوى التالى:

```
Dim x As Integer = 45
```

وإذا كان المتغير من نوع أحد التصنيفات، يمكننا التكوين الصريح لمثل من التصنيف الخاص بة عند الإعلان عنة باستخدام الكلمة المرشدة New ، كما يتضح من الكود التالى:

```
Dim x As New System.Windows.Forms.Form()
```

تعليمات التخصيص

تقوم تعليمات التخصيص (Assignment Statements) بتنفيذ عمليات تحديد القيم. وتقوم عملية التخصيص البسيطة بتخصيص القيمة التى على يمين عامل التخصيص للمتغير الذى على يسار ذلك العامل، كما يتضح مما يلى:

```
x = 42
```

فى المثال السابق، يتم تخصيص القيمة ٤٢ للمتغير x. ويمكن أن تكون القيمة التى على يمين عامل التخصيص قيمة حرفية، متغير، تعبير، أو استدعاء لدالة تعيد قيمة، كما فى المثال التالى :

```
x = y + z + MyFunction(3)
```

فى الكود السابق، يتم إضافة قيمة المتغير y إلى قيمة المتغير z إلى القيمة العائدة من تنفيذ الدالة Function(3) ثم يتم تخصيص قيمة هذا التعبير الإجمالية للمتغير x. ويمكن لعامل التخصيص أيضا، تخصيص قيم للمتغيرات من نوع سلسلة الرموز (String)، كما يتضح من المثال التالى:

```
Dim x As String
x = "String variable assignment"
x = "Con" & "cat" & "enation"
```

كما يستخدم عامل التخصيص أيضا فى وضع قيم منطقية بالمتغيرات المنطقية، كما يتضح من الكود التالى:

```
Dim x As Boolean
x = True
x = 45 > 1003
x = 45 > 1003 Or 45 > 17
```

ويستخدم عامل التخصيص أيضا فى ضبط خصائص الكائنات المختلفة، كما يتضح من

الكود التالى:

```
MyTextBox.Text = "This " & "is a " & "String"
```

وتستخدم العوامل المركبة فى إجراء عملية على قيمة متغير قبل إعادة تخصيص القيمة الجديدة لنفس المتغير. من أمثلة هذه العمليات ما يلى:

```
Dim x as Integer
x += 1
```

فى المثال السابق، يتم زيادة قيمة المتغير x بمقدار واحد ثم تخصيص القيمة الجديدة لنفس المتغير.

```
Dim x As String = "My "
x &= "String"    ' x equals "My String"
```

وفى المثال الأخير، يتم ربط السلسلة الموجودة فى المتغير x مع سلسلة رموز أخرى ثم تخصيص القيمة الجديدة لنفس المتغير. وإذا كان المتغير الذى على الجانب الأيسر من عامل تخصيص التسلسل (Concatenation) لا يحتوى على قيمة، يتم معاملته على أنة يحتوى على سلسلة فارغة ("").

التعليمات المنفذة (Executable Statements)

تقوم تعليمات التنفيذ (Executable Statements) ببدء عمل معين عن طريق تنفيذ وسيلة. ويمكن لهذه التعليمات تكرار التنفيذ فى حلقة أو التفرع خلال مجمعات من الكود. وتحتوى تعليمات التنفيذ فى الغالب على عوامل حسابية وعوامل شرطية.

يستخدم المثال التالى عبارة تعليمات If...Then..Else لتنفيذ مجمعات مختلفة من الكود بناءا على قيمة أحد المتغيرات. وداخل كل مجمع من الكود، تقوم عبارة تعليمات For..Next بتنفيذ مجموعة التعليمات التى بداخلها عدد محدد من المرات.

```
Public Sub StartWidget(aWidget As Widget, clockwise As Boolean, _
  revolutions As Integer)
  Dim counter As Integer
  If clockwise = True Then
    For counter = 1 to revolutions
      aWidget.SpinClockwise
    Next counter
  Else
    For counter = 1 to revolutions
      aWidget.SpinCounterClockwise
```

```
      Next counter
    End If
  End Sub
```

تقوم عبارة If...Then...Else فى المثال السابق باختبار قيمة المتغير clockwise. إذا كانت القيمة حقيقية (True)، يتم استدعاء وسيلة SpinClockwise. وعندما تكون القيمة غير حقيقية (False)، يتم استدعاء وسيلة SpinCounterClockwise. وتتسبب عبارات Next... For داخل كل مجمع من مجمعات الكود فى استدعاء الوسيلة المناسبة عدد من المرات يساوى قيمة المتغير revolutions.

الإجراءات

الإجراءات (Procedures) هى مجمعات من تعليمات Visual Basic داخل إطار يتكون من عبارة إعلان (Declaration Statement) وعبارة نهاية (End). ويتم استدعاء الإجراء من مكان أخر داخل الكود. وعند انتهاء تنفيذه، يعيد التحكم إلى الكود الذى قام باستدعائه. ويطلق على الكود الذى قام باستدعاء الإجراء، كود الاستدعاء (Calling Code). ويكون كود الاستدعاء فى صورة عبارة ، أو تعبير داخل عبارة يحدد الإجراء بالاسم وينقل التحكم فى التنفيذ إلية. ويستخدم Visual Basic عدة أنواع من الإجراءات:

- الإجراءات الفرعية (Sub)، التى تقوم بتنفيذ أفعال ولكنها لا تعيد قيمة إلى كود الاستدعاء.
- إجراءات معالجة الأحداث (Event-handling Procedures)، التى تعتبر من الإجراءات الفرعية التى يتم تنفيذها استجابة لأحد الأحداث التى تقع بسبب فعل المستخدم أو بواسطة البرنامج.
- إجراءات الدوال (Function Procedures)، التى تعيد قيمة إلى كود الاستدعاء.
- إجراءات الخصائص (Property Procedures)، التى تعيد وتحدد قيم لخصائص الكائنات والوحدات.

ويجب وضع كل سطر من الكود داخل أحد الإجراءات فى التطبيق. كما يجب تقسيم الإجراءات الكبيرة إلى إجراءات فرعية صغيرة لأن ذلك يجعل الكود أسهل فى القراءة والفهم.

وتفيد الإجراءات فى إنجاز المهام المتكررة أو المشتركة، مثل العمليات الحسابية المتكررة، معالجة النصوص وعمليات التحكم، وعمليات قواعد البيانات. ويمكن استدعاء إجراء من أماكن كثيرة مختلفة داخل الكود، ولهذا يمكن استخدام الإجراءات على أنها وحدات بناء للتطبيق. ويترتب على هيكلة الكود باستخدام الإجراءات تحقيق الفوائد التالية :

- تسمح الإجراءات بتقسيم البرامج إلى وحدات منطقية منفصلة، مما يسهل عملية البحث عن الأخطاء (Debugging).
- يمكن استخدام الإجراءات التى تم تكوينها فى أحد البرامج بواسطة البرامج الأخرى، بدون تعديل أو بتعديل قليل.

الإجراءات الفرعية

الإجراء الفرعي (Sub Procedure) هو سلسلة من تعليمات Visual Basic المحاطة بعبارة Sub وعبارة End Sub. وفى كل مرة يتم فيها استدعاء الإجراء، يتم تنفيذ التعليمات التى يحتوى عليها، بدءا من أول عبارة تنفيذية بعد عبارة Sub وانتهاء بعبارة End Sub، Exit Sub، أو Return. ويقوم إجراء Sub بتنفيذ أفعال ولكنة لا يعيد قيمة. ويمكن أن يتطلب الإجراء الفرعى إدخال معاملات، مثل الثوابت، المتغيرات، أو التعبيرات التى يتم تمريرها إلية بواسطة كود الاستدعاء. وتأخذ عبارة استدعاء الإجراء الفرعى، الصيغة التالية :

```
[accessibility] Sub subname[(argumentlist)]
 End Sub
```

يمكن أن يكون ترخيص الوصول Public، Protected، Friend، Protected Friend، أو Private. ويمكن تعريف الإجراءات الفرعية داخل وحدات الكود (Modules)، التصنيفات (Classes)، والهياكل (Structures). وهى عامة (Public) فى الأصل، مما يعنى إمكانية استدعائها من أى مكان فى التطبيق.

التركيب اللغوى لاستدعاء الإجراء الفرعى

يتم استدعاء الإجراء الفرعى صراحة باستخدام عبارة استدعاء مستقلة. ولا يمكن استدعاؤه بذكر اسمه داخل أحد التعبيرات. ويجب أن توفر عبارة الاستدعاء قيم للمعاملات

غير الاختيارية، كما يجب وضع قائمة المعاملات داخل قوسين. وإذا لم يكن هناك معاملات، يمكننا حذف الأقواس. كما أن استعمال كلمة الاستدعاء Call يكون اختياريا أيضا. ويأخذ التركيب اللغوى لاستدعاء الإجراء الصيغة التالية:

```
[Call] subname[(argumentlist)]
```

ولتوضيح استخدام الإجراءات الفرعية، نعرض المثال التالى الذى ينتج عنة عرض رسالة لإعلام مشغل الكمبيوتر المهمة التالية التى سوف يقوم بها التطبيق وتاريخ بدء المهمة. يشتمل المثال على إجراء TellOperator، الذى يقوم التطبيق باستدعائه فى بداية كل مهمة بدلا من إعادة كتابة الكود الذى يحقق ذلك فى كل مرة.

```
Sub TellOperator(ByVal Task As String)
   Dim Stamp As Date
   Stamp = TimeOfDay()
   MessageBox.Show("Starting " & Task & " at " & CStr(Stamp))
End Sub
```

ويمكن استدعاء الإجراء السابق من أى مكان فى التطبيق باستخدام الكود التالى:

```
Call TellOperator("file update")
```

إجراءات الدوال

يمكن تعريف إجراء الدالة (Function Procedure) بأنة سلسلة من تعليمات Visual Basic المحاطة بعبارة Function فى البداية وعبارة End Function فى النهاية. وفى كل مرة يجرى فيها استدعاء الإجراء، يتم تنفيذ التعليمات التى يحتوى عليها، بدءا من العبارة التى تلى عبارة Function وانتهاء بأول عبارة يتم مقابلتها من عبارات، End Function، Exit Function، أو Return. وهناك تشابه بين إجراء Function وبين إجراء Sub، والفرق بينهما هو أن إجراء Function يعيد إلينا قيمة. ويمكن استخدام المعاملات مع إجراء Function، مثل الثوابت، المتغيرات، أو التعبيرات. وللإعلان عن الدالة (Function)، نستخدم الصيغة التالية:

```
[accessibility] Function functionname[(argumentlist)] As datatype

End Function
```

ويمكن أن يكون ترخيص الوصول إلى الدالة Public، Protected، Friend، Protected Friend، أو Private. ويمكن تعريف إجراء الدالة فى وحدة (Module)، تصنيف (Class)،

أو هيكل بيانات (Structure). وترخيص الوصول إلية عام فى الأصل، مما يمكننا من استدعائه من أى موقع بالتطبيق. ويتم الإعلان عن معاملات إجراء Function بنفس طريقة الإعلان عن معاملات إجراء Sub.

القيم العائدة من الدوال

يطلق على القيمة التى يرسلها إجراء Function إلى كود الاستدعاء بعد انتهاء التنفيذ، القيمة العائدة (Return Value). وتعد الدوال القيم العائدة بإحدى طريقتين:

- تخصيص قيمة لاسم الدالة ذاتها فى عبارة أو أكثر من عبارات الإجراء. ولا يتم إعادة التحكم إلى كود الاستدعاء إلا بعد تنفيذ عبارة Exit Function أو عبارة End Function، كما يتضح من الكود التالى:

```
Function functionname[(argumentlist)] As datatype

  functionname = expression

End Function
```

- استخدام عبارة Return لتحديد القيمة العائدة، وإعادة التحكم إلى كود الاستدعاء فى نفس الوقت، كما يتضح من الكود التالى:

```
Function functionname[(argumentlist)] As datatype

  Return expression   'Control is returned immediately.

End Function
```

وترجع الفائدة فى تخصيص القيمة العائدة لاسم الدالة إلى أن التحكم لا يترك الدالة إلا بعد تنفيذ عبارة Exit Function أو End Function، مما يمكننا من تخصيص قيمة ابتدائية ثم تعديلها لاحقا عند الضرورة. وإذا كانت الدالة تعيد مصفوفة، فإننا لا نستطيع الوصول إلى عناصر المصفوفة داخل الدالة. على سبيل المثال، الكود التالى يؤدى إلى إطلاق رسالة خطأ :

```
Function AllOnes(ByVal N As Integer) As Integer()
  For I = 1 To N - 1
    AllOnes(I) = 1
  Next I
  Return AllOnes()
```

```
End Function
```

ويرتبط كل إجراء Function، مثل أى متغير، بنوع بيانات خاصة به يقرر نوع القيمة العائدة. ويتم تحديد نوع البيانات باستخدام فقرة As فى عبارة الإعلان عن الإجراء. الإعلانات التالية توضح ذلك:

```
Function Yesterday As Date
Function Sqrt(ByVal Number As Single) As Single
```

كود استدعاء الدالة

يمكن استدعاء إجراء الدالة بوضع اسم الدالة والمعاملات الخاصة بها إما على الجانب الأيمن فى عبارة تخصيص أو فى أحد التعبيرات، كما يتضح من كود الاستدعاء (Calling Syntax) التالى:

```
lvalue = functionname[(argumentlist)]
```

لتوضيح طرق استدعاء الدالة السابق ذكرها، نقوم بتكوين دالة rectangle التالى بيانها:

```
Function rectangle(ByVal length As Single, ByVal width As Single) As Single
    rectangle = (length + width) * 2
End Function
```

يمكن استدعاء هذه الدالة بوضع اسم الدالة على الجانب الأيمن فى عبارة تخصيص، كما يتضح من الكود التالى:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    Dim m As Single
    m = rectangle(12, 8)
    MessageBox.Show(CStr(m))
End Sub
```

ويمكن استدعاء الدالة يوضع اسمها فى عبارة، كما يتضح من الكود التالى:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    MessageBox.Show(CStr(rectangle(12, 8)))
End Sub
```

إجراءات الخصائص

الخاصية (Property) هى سلسلة من تعليمات Visual Basic، التى تقوم بالتعامل مع إعداد خاصية فى وحدة كود، تصنيف، أو هيكل بيانات. وتختلف الخاصية عن المتغير العام أو الحقل العام فى النواحى التالية:

- يتم تنفيذ الخاصية باستخدام الإجراءات الخاصة بها، وليس باستخدام عبارة إعلان عن الخاصية.
- يتم تنفيذ إجراءات الخصائص عند ضبط أو قراءة قيمة الخاصية. يمكننا ذلك من تنفيذ بعض الإجراءات عند محاولة كود عميل الوصول إلى الخاصية.
- لا يرتبط بإعلان الخاصية حجز أى جزء من الذاكرة. مع أن إجراءات Property تقوم غالبا، بتعريف متغيرات وثوابت محلية لا تكون متاحة أمام الكود العميل الذى يحاول الوصول إلى الخاصية.
- يمكن تحديد أن الخاصية تقبل القراءة فقط، تقبل الكتابة فقط، أو تقبل القراءة والكتابة.

ويقدم لنا Visual Basic اثنين من إجراءات الخصائص:

- إجراء Get، الذى يعيد قيمة الخاصية. ويتم استدعاء هذا الإجراء عند الوصول إلى الخاصية فى تعبير.
- إجراء Set، الذى يقوم بضبط قيمة الخاصية. ويتم استدعاء هذا الإجراء عند تخصيص قيمة للخاصية.

ويتم فى الغالب، تعريف إجراءات Property بصورة زوجية، باستخدام كلمات Get و Set، ولكن يمكن أن تشمل الخاصية على واحد من هذه الإجراءات فقط.

الإعلان عن الخاصية

يتم تعريف الخاصية باستخدام مجمع من الكود الذى يحيط به عبارة Property و عبارة End Property. وفى داخل مجمع كود الخاصية، تظهر إجراءات Property فى صورة مجمعات داخلية من الكود بين عبارة Get أو Set وبين عبارة End. الكود التالى يوضح صيغة الإعلان عن الخاصية والإجراءات التى تحتوى عليها:

```
[Default] [accessibility] Property propertyname[(argumentlist)] As datatype
Get
  Return expression
End Get
Set [(ByVal newvalue As datatype)]
  lvalue = newvalue
```

```
End Set
End Property
```

ويمكن أن يكون ترخيص الوصول Public، Protected، Friend، Protected Friend، أو Private. وترخيص الوصول الإفتراضى هو Public.

كود الاستدعاء (Calling Syntax)

يتم استدعاء أجراء الخاصية ضمنيا بواسطة الكود الذى يستخدم الخاصية. ويستخدم الكود اسم الخاصية بنفس الطريقة التى يستخدم بها اسم متغير، فيما عدا أنة يجب إدخال قيم للمعاملات غير الاختيارية، كما يجب وضع قائمة المعاملات بين أقواس. وفى حالة عدم وجود معاملات، يمكن حذف الأقواس. والكود اللازم للاستدعاء الضمنى لإجراء Set Property يأخذ الصيغة التالية:

```
propertyname[(argumentlist)] = expression
```

بينما يأخذ كود استدعاء إجراء Get Property إحدى الصيغ التالية:

```
lvalue = propertyname[(argumentlist)]
Do While (propertyname[(argumentlist)] > expression)
```

معاملات الإجراءات

يحتاج الإجراء فى معظم الحالات، إلى بعض المعلومات عن الظروف التى يتم استدعاؤه من خلالها. والإجراء الذى يقوم بتنفيذ مهام متكررة أو مشتركة يتطلب معلومات مختلفة مع كل استدعاء. تتكون هذه المعلومات من متغيرات، ثوابت، وتعبيرات يجرى تمريرها إلى الإجراء عند استدعائه. ويطلق على كل قيمة يجرى تمريرها إلى إجراء مصطلح معامل (argument).

الإعلان عن نوع بيانات المعامل

نقوم بالإعلان عن نوع بيانات لأحد المعاملات باستخدام فقرة As عند الإعلان عنها. على سبيل المثال، الدالة التالية، تقبل سلسلة رموز (String) و رقم صحيح (Integer):

```
Function Appointment(ByVal Day As String, ByVal Hour As Integer) As String
End Function
```

وعندما تكون عبارة Option Strict غير فعالة، يصبح استخدام فقرة As اختياريا، إلا أنها تصبح واجبة الاستعمال إذا قام أحد المعاملات باستعمالها. وعلى العكس من ذلك، تصبح فقرة As مطلوبة لجميع المعاملات عندما يكون خيار Option Strict فعالا.

تمرير المعامل بالقيمة (Passing By Val)

عند تمرير معامل متغير بالقيمة، باستخدام كلمة ByVal، لا يستطيع الإجراء تعديل المتغير الأصلى فى كود الاستدعاء. من ناحية أخرى، إذا كان المعامل من نوع متغيرات المراجع (Reference Type)، يمكننا تغيير قيمة المتغير الذى تجرى الإشارة إلية. على سبيل المثال، إذا كان المعامل متغير مصفوفة، يمكننا تغيير قيمة عنصر أو أكثر من عناصر المصفوفة، ولكن لا يمكننا تخصيص مصفوفة جديدة للمتغير. وينعكس التغيير الذى يحدثه الإجراء فى عناصر المصفوفة بكود الاستدعاء.

يعرض المثال التالى، اثنين من الإجراءات الفرعية التى يتم تمرير متغير مصفوفة بالقيمة إلى كل منهما والعمل على عناصر هذه المصفوفات داخل الإجراءات. يقوم إجراء increase بإضافة واحد صحيح إلى قيمة كل عنصر من عناصر المصفوفة. ويقوم إجراء Replace بتخصيص مصفوفة جديدة لمصفوفة المعامل A() ثم يضيف واحد إلى عناصر المصفوفة بعد ذلك. ومع ذلك، لا يؤثر إعادة التخصيص على متغير المصفوفة فى كود الاستدعاء.

```
Public Sub Increase(ByVal A() As Long)
    Dim J As Integer
    For J = 0 To UBound(A)
      A(J) = A(J) + 1
    Next J
End Sub

Public Sub Replace(ByVal A() As Long)
   Dim J As Integer
   Dim K() As Long = {100, 200, 300}
   A = K
   For J = 0 To UBound(A)
     A(J) = A(J) + 1
   Next J
End Sub
```

ولاستدعاء الإجراءات السابقة، نستخدم كود الاستدعاء التالى:

```
Dim N() As Long = {10, 20, 30, 40}
Increase(N)
Replace(N)
```

تمرير المعاملات بالمرجع

عند تمرير مرجع متغير باستخدام كلمة ByRef، يمكن للإجراء تعديل المتغير ذاته. وبصفة خاصة، عندما يكون المتغير أحد الكائنات، يمكن تخصيص متغير جديد له. هذا التخصيص الجديد يؤثر على المتغير فى كود الاستدعاء.

على سبيل المثال، عندما يتم تمرير المصفوفة بالمرجع فى إجراء Replace السابق، سوف يتم تخصيص مصفوفة ()Kالجديدة لمتغير المصفوفة N فى كود الاستدعاء. وسوف يترتب على ذلك تغيير عناصر المصفوفة فى كود الاستدعاء.

آلية تمرير المعاملات

عند تمرير معامل من غير المتغيرات، لايمكن للإجراء تعديل هذا المعامل فى كود الاستدعاء، سواء تم تمريرة بالقيمة أو بالمرجع. وبالنسبة للمتغيرات التى يتم تمريرها فى صورة معاملات إلى الإجراءات، تلخص النقاط التالية العلاقة بين نوع بيانات هذه المتغيرات وبين آلية تمرير المعاملات.

- عندما يحتوى المتغير على قيمة، لا يستطيع الإجراء تغييره أو تغيير أى من أعضائه عند التمرير بالقيمة (ByVal). وفى حالة التمرير بالمرجع (ByRef)، يمكن للإجراء تغيير المتغير وأعضائه.
- عندما يحتوى المتغير على مرجع (Reference)، لايمكن للإجراء تغيير المتغير ولكن يمكنة تغيير أعضاء المثل (Instance Members) الذى يشير إلية عند التمرير بالقيمة. وعند التمرير بالمرجع، يمكن للإجراء تغيير المتغير وتغيير أعضاء المثل الذى يشير إلية هذا المتغير.

الإعلان عن الإجراء هو الذى يحدد آلية تمرير المعاملات. ولا يستطيع كود الاستدعاء تغيير آلية ByVal، ولكن فى حالة التمرير بالمرجع، يمكن لكود الاستدعاء فرض آلية التمرير بالقيمة عن طريق إحاطة اسم المعامل بالأقواس عند استدعاء الإجراء. الكود التالى، يوضح كيفية فرض آلية التمرير بالقيمة :

```
Sub DoSomething(ByRef InString As String)

End Sub

Call DoSomething((Str))
```

فى الكود السابق، تم الإعلان عن إجراء DoSomething، الذى يتم تمرير معامل من نوع String إلية بالمرجع. لفرض التمرير بالقيمة، نضع المرجع بين قوسين فى قائمة معاملات الإجراء، كما هو موضح بعبارة استدعاء الدالة. والوضع الإفتراضى فى Visual Basic هو التمرير بالقيمة. ومع ذلك، يمكن جعل الكود أكثر وضوحا باستخدام كلمة ByVal أيضا. كما يتضح من الكود التالى:

```
Sub PostAccounts(ByVal AcctNum As Integer)

End Sub
```

وعند الرغبة فى تمرير متغير بالمرجع، يجب استخدام كلمة ByRef لتحديد استخدام هذه الآلية. يمكن إيضاح ذلك باستخدام الكود السابق بالصورة التالية:

```
Sub PostAccount(ByVal AcctNum As Integer, ByRef RunningTotal As Single)

End Sub
```

وترجع أهمية استخدام كلمة ByRef مع المعامل إلى أنها تتيح لنا إعادة قيمة إلى كود الاستدعاء من خلال المعامل. وفائدة تمرير معامل بالقيمة هى المحافظة على متغير كود الاستدعاء من التغيير.

تمرير الاجراء بالموقع والاسم

عند استدعاء إجراء Sub أو إجراء Function، يمكننا تمرير المعاملات بالموقع، بمعنى تمريرها طبقا لترتيب ظهورها فى تعريف الإجراء. كما يمكن تمرير المعاملات بأسمائها بغض النظر عن مواقعها. وعند تمرير معامل بالاسم، نحدد اسم المعامل متبوعا بعلامة الوقف الإستدراكى وإشارة التساوي (=:)، متبوعة بقيمة المعامل. ويمكن إدخال أسماء المعاملات بأي ترتيب. على سبيل المثال، الإجراء التالى يأخذ ثلاثة معاملات:

```
Sub StudentInfo(ByVal Name As String, Optional ByVal Age As Short = 0, _
                Optional ByVal Birth As Date = #1/1/2000#)
    Debug.WriteLine(Name, Age, Birth)
End Sub
```

عند استدعاء هذا الإجراء، يمكننا إدخال المعاملات بالموقع، بالاسم، أو باستخدام مزيج من الطريقتين. استدعاء الإجراء السابق باستخدام مواقع المعاملات يأخذ الصيغة

التالية :

```
StudentInfo("Mary", 19, #21-Sep-1981#)
```

وعند حذف أحد المعاملات الاختيارية من قائمة معاملات أحد الإجراءات، يجب الاحتفاظ بموقع هذا المعامل باستخدام الفاصلة، كما يتضح من الكود التالى:

```
StudentInfo("Mary", , #21-Sep-1981#)
```

وعند تمرير المعاملات بالاسم، يأخذ كود استدعاء الإجراء السابق، الصيغة التالية:

```
StudentInfo(Age:=19, Birth:=#21 Sep 1981#, Name:="Mary")
```

ويمكن المزج بين تمرير المعاملات بالاسم وتمرير المعاملات بالموقع، كما يتضح من الكود التالى:

```
StudentInfo("Mary", Birth:=#21 Sep 1981#)
```

وعند تمرير المعاملات بالاسم، يمكن حذف المعاملات الاختيارية فقط. كما أننا لا نستطيع تمرير معامل مصفوفة بالاسم. ولتحديد أن المعامل اختياريا، نضع كلمة Optional قبل اسم المعامل وكلمة تعريف آلية تمريرة فى عبارة تعريف الإجراء. كما يجب تحديد قيمة المعامل الاختياري فى عبارة التعريف. الكود التالى يوضح صيغة الإعلان عن إجراء به معاملات اختيارية :

Sub *subname*(ByVal *arg1* As *type1*, Optional ByVal *arg2* As *type2* = *default*)

استخدام المصفوفات فى المعاملات

لايمكن استدعاء إجراء باستخدام معاملات أكثر من المعاملات المحددة فى تعريف الإجراء. وعند الحاجة إلى استخدام عدد غير محدد من المعاملات، يمكن الإعلان عن معامل من نوع المصفوفة لكى يمكن للإجراء قبول مصفوفة من القيم فى أحد المعاملات. وليس من الضروري معرفة عدد العناصر فى مصفوفة المعامل عند تعريف الإجراء. حيث يتم تحديد حجم المصفوفة فى كل استدعاء منفرد للإجراء. ولتعريف أحد المعاملات على أنة مصفوفة، نستخدم الكلمة الإرشادية ParamArray للإشارة الى أن المعامل مصفوفة. وهناك عدد من القواعد التى تحكم استخدام المصفوفات فى المعاملات، نعرضها فى النقاط التالية :

- يمكن أن يكون بالإجراء معامل مصفوفة واحد، ويجب أن يكون أخر معامل فى تعريف الإجراء.
- يجب تمرير معامل المصفوفة بالقيمة. ومن المفضل استخدام كلمة ByVal للدلالة

على ذلك.

- يجب أن يعامل كود الإجراء معامل المصفوفة على أنة مصفوفة ذات بعد واحد، ونوع بيانات كل عنصر من عناصرها يماثل نوع بيانات معامل المصفوفة.
- ويعتبر معامل المصفوفة اختياريا بصفة تلقائية. وقيمته الافتراضية عبارة عن مصفوفة خالية ذات بعد واحد ونوع بيانات عناصرها يماثل نوع بيانات معامل المصفوفة.
- كل المعاملات السابقة على معامل المصفوفة، يجب ألا تكون اختيارية. ويجب أن يكون معامل المصفوفة هو المعامل الاختياري الوحيد.

وعند استدعاء إجراء به معامل مصفوفة، يمكننا تمرير أى من الأتي فى معامل المصفوفة:

- حذف معامل المصفوفة. مما يترتب علية تمرير مصفوفة خالية إلى الإجراء. ويمكن أيضا تمرير كلمة Nothing للحصول على نفس النتيجة.
- قائمة بعدد غير محدد من المعاملات، مفصولة بينها بالفاصلة. ويكون نوع بيانات كل معامل قابل للتحويل ضمنيا إلى نوع بيانات معامل المصفوفة.
- مصفوفة يكون نوع بيانات عناصرها من نفس نوع بيانات معامل المصفوفة.
- المثال التالى، يوضح كيفية تعريف إجراء به معامل مصفوفة:

```
    Sub StudentScores(ByVal Name As String, ByVal ParamArray Scores() As String)
        Dim I As Integer
        Debug.WriteLine("Scores for " & Name & ":")
        For I = 0 To UBound(Scores)
            Debug.WriteLine("Score " & I & ": " & Scores(I))
        Next I
    End Sub
```

ويمكن استدعاء الإجراء السابق باستخدام الكود التالى:

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
        Dim A() As String = {"24", "34", "56"}
        StudentScores("Hassan", A)
    End Sub
```

التحميل الزائد للإجراءات

المقصود بالتحميل الزائد لإجراء (Overloading a Procedure) هو تعريف نسخ منة باستخدام نفس الاسم ولكن مع قائمة معاملات مختلفة. يترتب على ذلك، أن التفرقة بين هذه النسخ يكون بقائمة المعاملات وليس بالاسم.

القواعد التى تحكم التحميل الزائد للإجراءات

عند القيام بالتحميل الزائد لأحد الإجراءات، تنطبق القواعد التالية:

- كل نسخة من الإجراء يجب أن يكون لها نفس اسم الإجراء.
- كل نسخة يجب أن تختلف عن النسخ الأخرى فى واحد على الأقل من الموضوعات التالية:
  - عدد المعاملات
  - ترتيب المعاملات
  - أنواع بيانات المعاملات

يستخدم برنامج ترجمة الكود اسم الإجراء وقائمة المعاملات للتحقق من أن الاستدعاء يتوافق مع التعريف. وهذه البنود يطلق عليها شارة الإجراء(Signature of the Procedure). لايمكن تكوين نسخ معدلة من إجراء بدون إحداث تغييرات فى قائمة المعاملات. وبصفة خاصة، لايمكن تكوين نسخ معدلة من إجراء عن طريق تغيير واحد أو أكثر من البنود التالية:

- الكلمات الخاصة بتراخيص الوصول إلى الإجراء (Procedure Modifier)، مثل Public، Shared.
- أسماء المعاملات.
- كلمات الوصول إلى المعاملات، مثل ByRef، Optional.
- نوع بيانات القيمة العائدة.

النسخ المعدلة المتعددة من إجراء

نفترض أننا نريد ترحيل عملية إلى رصيد حساب أحد العملاء، ونريد فى نفس الوقت

الوصول إلى هذا العميل إما باسم العميل أو برقم الحساب. يمكننا إنجاز ذلك، بتعريف إجراءين مختلفين من إجراءات Sub، كما يتضح من الكود التالى بيانه:

```
Sub PostName(ByVal CustName As String, ByVal Amount As Single)
Sub PostAcct(ByVal CustAcct As Integer, ByVal Amount As Single)
```

الطريقة البديلة لتنفيذ ذلك هى تكوين نسخة معدلة من إجراء. ويتم ذلك باستخدام كلمة Overloads لتعريف نسخة معدلة من الإجراء لكل قائمة معاملات مختلفة، كما يتضح مما يلى:

```
Overloads Sub Post(ByVal CustName As String, ByVal Amount As Single)
 MessageBox.Show("Customer " & CustName & " has been updated by " &
Amount)
End Sub
Overloads Sub Post(ByVal CustAcct As Integer, ByVal Amount As Single)
MessageBox.Show("Customer " & CustAcct & " has been updated by " &
Amount)
End Sub
```

وعند الرغبة فى قبول مبلغ من نوع Decimal أو Single، يمكن تعريف نسخ أخرى من إجراء Post للسماح بهذه التعديلات. فإذا تم تنفيذ ذلك لكل إجراء من الإجراءين السابقين، فإننا نحصل على أربعة نسخ من إجراء Post، جميعها لها نفس الاسم ولكن كل منها يحتوى على قائمة معاملات مختلفة. والهدف من تكوين نسخ معدلة من إجراء هو تحقيق المرونة فى استدعاء هذه الإجراءات. على سبيل المثال، لاستخدام إجراء Post الذى جرى الإعلان عنة فى الكود السابق، يمكن الحصول على العميل فى صورة سلسلة رموز أو رقم صحيح، ثم استدعاء نفس الإجراء فى كل من الحالتين، كما يتضح من الكود التالى، الذى يقوم باستدعاء إجراء Post السابق:

```
Imports MSVB = Microsoft.VisualBasic
 Private Sub Update_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Update.Click
    Dim Customer As String
    Dim AccountNum As Integer
    Dim Amount As Single
    Customer = MSVB.Interaction.InputBox("Enter customer name or
number")
    Amount = MSVB.Interaction.InputBox("Enter transaction amount")
    Try
      AccountNum = CInt(Customer)
```

```
            Post(AccountNum, Amount)
        Catch
            Post(Customer, Amount)
        End Try
    End Sub
```

قيود التحميل الزائد للإجراءات التى تحتوى على معاملات اختيارية

يساوى الإجراء الذى يحتوى على معامل اختياري نسختين معدلتين من نفس الإجراء، واحدة بها قائمة معاملات تحتوى على المعامل الاختياري والنسخة الأخرى بها قائمة معاملات لا تحتوى على المعامل الاختياري. لهذا السبب، لا نستطيع تكوين نسخ معدلة من هذا الإجراء تحتوى على قائمة معاملات تساوى أى من هاتين القائمتين. الإعلان التالى يوضح ذلك:

```
Overloads Sub Q(ByVal B As Byte, Optional ByVal J As Long = 6)
```

التعريف السابق، يساوى التعريفيين التاليين:

```
Overloads Sub Q(ByVal B As Byte)
Overloads Sub Q(ByVal B As Byte, ByVal J As Long)
```

ولذلك تكون النسخة المعدلة التالية غير صالحة:

```
Overloads Sub Q(ByVal C As Byte, ByVal K As Long)
```

ولكن النسخة التالية تستخدم قائمة معاملات مختلفة، ولهذا تعتبر نسخة معدلة صالحة:

```
Overloads Sub Q(ByVal B As Byte, ByVal J As Long, ByVal S As Single)
```

التحكم فى تنفيذ البرامج

تقوم البرامج بتنفيذ التعليمات التى تحتوى عليها بالترتيب من البداية إلى النهاية. وهناك بعض البرامج التى تستخدم هذا النموذج المبسط فقط. غير أن معظم قوة وخدمات لغات البرمجة تكمن فى القدرة على تغيير مسار التنفيذ باستخدام تعليمات التحكم (Control Statements) وحلقات التنفيذ (Loops). تقوم تعليمات التحكم بتنظيم تدفق تنفيذ التعليمات، مما يسمح باتخاذ قرارات تتعلق بتنفيذ أو عدم تنفيذ بعض التعليمات. وتمكننا حلقات التنفيذ من تكرار تنفيذ التعليمات بدلا من تنفيذها لمرة واحدة.

هياكل القرارات

يسمح لنا Visual Basic باختبار الشروط وتنفيذ عمليات مختلفة بناءا على نتائج هذا الاختبار. يمكننا اختبار تحقق الشرط أو عدم تحققة، بالنسبة لقيم متنوعة خاصة بأحد التعبيرات، أو بالنسبة للإستثناءات المتنوعة التى تتولد عند تنفيذ سلسلة من التعليمات. ويدعم Visual Basic عدد من تعليمات اتخاذ القرارات التى تتمثل فيما يلى:

- If...Then
- If...Then...Else
- Select...Case
- Try...Catch...Finally

تعليمات If...Then...Else

يمكن استخدام تعليمات If...Then...Else لتنفيذ عبارة معينة أو مجمع من التعليمات بناءا على القيمة المنطقية للشرط. وتنتج الشروط دائما من المقارنة بين قيمتين، غير أن الشرط يمكن أن يكون أى تعبير له قيمة منطقية (True أو False).

تنفيذ التعليمات عند تحقق الشروط

لتنفيذ عبارة واحدة عند تحقق أحد الشروط، يمكننا استخدام التركيب اللغوى لعبارة If...Then...Else على سطر واحد مع حذف عبارات Else و End If، كما يتضح من الكود التالى:

```
Sub FixDate()
  Dim MyDate As Date = #2/13/1973#
  If MyDate < Now Then MyDate = Now
End Sub
```

ولتنفيذ أكثر من سطر من الكود عندما يكون الشرط حقيقيا، نستخدم التركيب اللغوى لعبارة If الذى يشمل عدة سطور، ويحتوى على عبارة End If. وعندما لا يكون هناك كود يجب تنفيذه عندما يكون الشرط غير حقيقي، لا نستخدم عبارة Else، كما يتضح من الكود التالى:

```
Sub ShowDate(ByVal MyDate As Date)
      If MyDate < Today Then
         Debug.Write("The date is earlier than today ")
         Debug.Write("But tomorrow is another day")
      End If
```

```
End Sub
```

تنفيذ بعض التعليمات عند توفر بعض الشروط وتنفيذ تعليمات أخرى عند عدم توفر هذه الشروط

ويمكن استخدام عبارة If...Then...Else لتحديد مجموعتين من التعليمات. ويتم تنفيذ أحد المجموعتين عند تحقق الشرط، وتنفيذ المجموعة الأخرى عندما لا يتحقق الشرط، كما يتضح من المثال التالى:

```
Sub ShowDate(ByVal MyDate As Date)
    If MyDate < Today Then
       Debug.Write("The date is earlier than today ")
    Else
       Debug.Write("The date either is today or after today")
    End If
End Sub
```

اختبار شروط إضافية عندما يكون الشرط الأول غير حقيقى

يمكننا إضافة عبارة أو أكثر من عبارات ElseIf إلى عبارة If...Then...Else لاختبار شروط إضافية عندما يكون الشرط الأول غير حقيقى. على سبيل المثال، الدالة التالية تقوم بحساب علاوات الأجور بناءا على معدل الأداء. ويتم تنفيذ العبارة التى تلي عبارة Else عندما تكون الشروط فى جميع عبارات If و ElseIf غير حقيقية.

```
Function Bonus(ByVal Performance As Single, ByVal Salary As Decimal) _
  As Decimal
  If Performance = 1 Then
    Return Salary * 0.1
  ElseIf Performance = 2 Then
    Return Salary * 0.09
  ElseIf Performance = 3 Then
    Return Salary * 0.07
  Else
    Return 0
  End If
End Function
```

ويقوم Visual Basic باختبار الشروط بنفس ترتيب ظهورها فى عبارات If...Then...Else إلى أن يتم تحقق أحد الشروط أو مقابلة عبارة Else، وفى أى من الحالتين يقوم بتنفيذ مجمع التعليمات ذات العلاقة. بعد ذلك، يتفرع التنفيذ إلى نهاية مجمع كود

If...Then...Else. ويمكن أن يكون لدينا أى عدد من عبارات ElseIf، أو لاشيء منها. ويمكن إدراج أو حذف عبارة Else بغض النظر عن وجود عبارات ElseIf. ويمكن تداخل عبارات If...Then...Else بعدد المستويات التى نريدها. غير أن استخدام عبارات Select...Case فى هذه الحالة، يجعل الكود أكثر قابلية للفهم عند القراءة.

تعليمات Select...Case

عند الحاجة إلى مقارنة نفس التعبير مع قيم عديدة مختلفة، يمكننا استخدام عبارات Select...Case بدلا من عبارات If...Then...Else. وبينما تستطيع عبارات If و ElseIf تقييم تعبير مختلف فى كل عبارة، فإن عبارة Select تقيم تعبير واحد مرة واحدة وتستخدم القيمة الناتجة فى كل مقارنة. ويقوم Visual Basic بمقارنة قيمة التعبير مع القيم المذكورة فى عبارات Case بترتيب ظهورها فى مجمع Select...Case. وعند وجود تناظر أو عبارة Case Else، يتم تنفيذ مجمع التعليمات ذات العلاقة. ثم يتم بعد ذلك تنفيذ الكود الذى يلى عبارة End Select فى جميع الحالات. ويمكن أن يكون لدينا أى عدد من عبارات Case، ويمكن وضع عبارة Case Else أو حذفها من الكود.

فى المثال التالى، يتم استخدام Select...Case لتقييم معدلات الأداء الذى يتم تمريرة إلى إجراء Function. ومن الملاحظ أن كل عبارة Case يمكن أن تحتوى على أكثر من قيمة، نطاق من القيم، أو تركيب من القيم وعوامل المقارنة. وعندما تحتوى عبارة Case على أكثر من قيمة، يتم تنفيذ مجمع تعليمات Case إذا تساوت أى من القيم مع قيمة تعبير Select.

```
Function Bonus(ByVal Performance As Single, ByVal Salary As Decimal) _
    As Decimal
  Select Performance
    Case 1
      Return Salary * 0.1
    Case 2, 3
      Return Salary * 0.09
    Case 5 To 7
      Return Salary * 0.07
    Case 4, 8 To 10
      Return Salary * 0.05
    Case Is < 15
      Return 100
    Case Else
```

```
        Return 0
    End Select
End Function
```

تعليمات Try...Catch...Finally

يمكن استخدام عبارات Try...Catch...Finally لمعالجة هياكل الاستثناءات. يسمح لـ
ذلك بتنفيذ مجمع تعليمات معين عند حدوث استثناء محدد أثناء تشغيل الكود. وعنده
يحدث ذلك، فإن التعبير المستخدم هو قيام النظام بقذف استثناء (Throw the exception)
وقيام البرنامج باصطياد (Catching) هذا الاستثناء. وينتقل التنفيذ إلى داخل مجمع
Try...Catch...Finally بدءا بالتعليمات التي تلي عبارة Try. فإذا قام Visual Basic باستكمال
تنفيذ التعليمات بدون أخطاء، فإنه يتحقق من وجود عبارة Finally الاختيارية في النهاية
إذا كانت هناك عبارة Finally، يقوم Visual Basic بتنفيذ مجمع التعليمات الخاص بها
وفي جميع الأحوال، ينتقل التحكم في التنفيذ بعد ذلك إلى العبارة التالية لعبارة End Try.

وعند حدوث استثناء، يقوم Visual Basic بفحص تعليمات Catch بترتيب ظهورها
فإذا وجد أحد التعليمات التي تعالج الاستثناء الناتج، فانه يقوم بتنفيذ تلك التعليمات
وعند الانتهاء من تنفيذ مجمع تعليمات Catch، يقوم بتنفيذ مجمع تعليمات Finally عنـ
وجودة. ينتقل التنفيذ بعد ذلك إلى العبارة التى تلى عبارة End Try. وتقوم عبارة atch
بالتعامل مع الاستثناءات التى من نفس النوع المعلن عنة فيها أو نوع مشتق منة. فإذا تـ
إدخال تعليمات في مجمع Catch، لا تحدد نوع التنفيذ، فإنه يتم استخدامها في معالجـ
أى استثناء مشتق من تصنيف Exception. وإذا كانت هذه التعليمات هى أخر التعليمات
في مجمع تعليمات Catch، فإنها تعالج أى استثناء لم يتم تحديد تعليمات معالجته.

وسواء تم تحديد نوع الاستثناء أو لم يتم، فإن عبارة Catch يمكن أن تشتمل على فقر
When بصفة اختيارية. تتكون عبارة When من تعبير ينتج عنة قيمة منطقية تتحكم فـ
تنفيذ كود معالجة الاستثناء. حيث يتم معالجة الاستثناء فقط عندما تكون قيمة عبارة hen
حقيقية (True). وعندما تكون هناك عبارة Finally، فإن مجمع التعليمات الخاص بـ
يكون دائما أخر ما يتم تنفيذه قبل أن يترك التحكم Try...Catch...Finally. ويعتبر ذلـ
حقيقيا حتى عندما يحدث استثناء لم يتم معالجته، أو عند تنفيذ عبارة Exit Try. ويمكـ
أن يكون لدينا أى عدد من عبارات Catch، كما يمكن أن توجد أو لا توجد عبارة inally

وفى جميع الأحوال، يجب أن يكون لدينا على الأقل عبارة Catch أو عبارة Finally واحدة.

فى المثال التالى، تحاول عبارة Try...Catch...Finally أن تحسب التاريخ والوقت بعد مائة سنة من التاريخ الذى يجرى إدخاله فى متغير كائن باسم GivenDate:

```
Dim GivenDate As Object
Dim NextCentury As Date
Try
  NextCentury = DateAdd("yyyy", 100, GivenDate)
  Catch ThisExcep As System.ArgumentException
  Catch ThisExcep As ArgumentOutOfRangeException
  Catch ThisExcep As InvalidCastException
  Catch
  Finally
End Try
```

فى هذا المثال، يمكن معالجة الاستثناءات المتوقعة عند تنفيذ دالة DateAdd فى عبارات Catch الثلاثة الأول، ويمكن التعامل مع أى استثناء غير متوقع فى عبارة Catch الأخيرة. وبغض النظر عما يحدث، يتم تنفيذ مجمع تعليمات Finally قبل ترك عبارة Try...Catch...Finally. وعندما لا يتم تعريف استثناء ThisExcep باستخدام عبارة إعلان عنة، مثل Dim، فإن عبارة Catch مع فقرة As تعتبر إعلانا عن متغير الاستثناء.

ولإيضاح استخدام عبارة Catch، نعرض المثال التالى الذى يقوم بتنفيذ الخطوات التالية:

١. استقبال النص الذى يدخلة المستخدم فى مربع نص txtNumerator ثم تحويلة إلى قيمة عددية وتخزينها فى المتغير numerator.

٢. استقبال النص الذى يدخلة المستخدم فى مربع نص txtDenominator ثم تحويلة إلى قيمة عددية وتخزينها فى المتغير denominator.

٣. قسمة المتغير numerator على المتغير denominator وعرض النتيجة فى متحكم Label.

٤. إذا كانت قيمة المتغير denominator تساوى صفر، يطلق البرنامج رسالة خطأ استثنائي.

٥. تقوم عبارة Catch فى مجمع كود Try...Catch...Finally باصطياد الخطأ وعرض رسالة باستخدام خاصية Message فى تصنيف DivideByZeroException.

يوضح الإجراء التالى كيفية تنفيذ الخطوات السابقة باستخدام الكود:

```
Private Sub performDivision()
    Dim numerator As Integer = CInt(txtNumerator.Text)
    Dim denominator As Integer = CInt(txtDenominator.Text)
    Try
        lblResult.Text = CStr(numerator \ denominator)
    Catch zerodivide As DivideByZeroException
        MessageBox.Show(zerodivide.Message)
    End Try
End Sub
```

هياكل الحلقات

تسمح لنا هياكل الحلقات بتنفيذ التعليمات التى بداخلها بالتكرار. ويستمر تكرار التنفيذ إلى أن يصبح أحد الشروط حقيقيا أو غير حقيقي، عدد من المرات، أو مرة واحدة لكل كائن فى مجموعة. وتشتمل هياكل الحلقات التى يدعمها Visual Basic على ما يلى:

- While
- Do...Loop
- For...Next
- For Each...Next

حلقة While

يمكن استخدام حلقة While لتنفيذ مجمع من التعليمات لعدد لانهائي من المرات بناءا على القيمة المنطقية لأحد الشروط. ويتم تكرار تنفيذ التعليمات طالما استمر الشرط حقيقيا. وينتج الشرط فى العادة من المقارنة بين قيمتين، ولكنة يمكن أن يكون أى تعبير يقبل التحول إلى قيمة منطقية (True أو False). وتقوم عبارة While دائما باختبار الشرط قبل بدء تنفيذ الحلقة.

فى المثال التالى، يقوم إجراء CheckWhile باختبار الشرط قبل الدخول فى حلقة التنفيذ. فإذا كانت القيمة الابتدائية للمتغير Number تساوى ٦ بدلا من ١٠، فإن التعليمات التى بداخل الحلقة لن يتم تنفيذها مطلقا.

```
Sub CheckWhile()
```

```
    Dim Counter As Integer = 0
    Dim Number As Integer = 10
    While Number > 6
      Number = Number - 1
      Counter = Counter + 1
    End While
    MsgBox("The loop ran " & Counter & " times.")   ' Runs 4 times.
End Sub
```

ونستخدم عبارة Exit While للخروج من حلقة While عند تحقق أحد الشروط التى تستدعى إيقاف التنفيذ، مثل استمرار تنفيذ الحلقة إلى ما لانهاية أو حدوث أخطاء. عندما يكون الشرط المذكور حقيقيا، نستخدم عبارة Exit While للهروب، وإذا كان الشرط غير حقيقي، نستمر فى تنفيذ التعليمات داخل الحلقة. فى المثال التالى، هناك شرطان، الشرط الأول هو أن المتغير Number لا يساوى 10 لكى يستمر تنفيذ حلقة التعليمات، والشرط الثانى هو وصول قيمة المتغير Number إلى أقل من الصفر لكى يتم الخروج من الحلقة. بالنظر إلى أن المتغير المذكور يبدأ بالقيمة 8، لذا سوف تستمر الحلقة إلى ما لانهاية. وللخروج من الحلقة اللانهائية، نستخدم الشرط الثانى من خلال عبارة If لاختبار قيمة المتغير Number والخروج من الحلقة عندما تصبح قيمته أقل من صفر، مما يمنع التنفيذ اللانهائي للتعليمات الموجودة داخل الحلقة.

```
Sub ExitWhileExample()
    Dim Counter As Integer = 0
    Dim Number As Integer = 8
    While Number <> 10
      If Number < 0 Then Exit While
      Number = Number - 1
      Counter = Counter + 1
    End While
    MsgBox("The loop ran " & Counter & " times.")
End Sub
```

طلقات Do...Loop

يمكننا استخدام عبارات Do...Loop لتنفيذ مجمع من التعليمات لعدد غير محدد من المرات بناءا على القيمة المنطقية للشرط المستخدم. ويمكن استمرار تنفيذ الحلقة عندما يكون الشرط حقيقيا أو إلى أن يصبح حقيقيا. وتنتج قيمة الشرط فى الغالب من عملية المقارنة بين

قيمتين، ولكنة يمكن أن يكون أى تعبير يقبل التحول إلى قيمة منطقية.

تكرار تنفيذ التعليمات إثناء تحقق الشرط

هناك طريقتان لاستخدام كلمة While لاختبار أحد الشروط فى حلقة Do. يمكننا اختبار الشرط قبل الدخول فى الحلقة، أو اختباره بعد تشغيل الحلقة لمرة واحدة على الأقل. ويستمر التنفيذ المتكرر طالما بقى الشرط حقيقيا.

فى المثال التالى، يقوم إجراء CheckWhileFirst باختبار الشرط قبل الدخول فى حلقة التنفيذ. فإذا تم جعل القيمة الابتدائية للمتغير Number تساوى ٦ بدلا من ١٠، فإن التعليمات الموجودة داخل الحلقة لن يتم تنفيذها مطلقا. وفى إجراء CheckWhileLast، يتم تنفيذ التعليمات داخل الحلقة مرة واحدة على الأقل قبل اختبار الشرط، الذى يكون غير حقيقى عند القيام بالاختبار الأول.

```
Sub CheckWhileFirst()
  Dim Counter As Integer = 0
  Dim Number As Integer = 10
  Do While Number > 6
    Number = Number - 1
    Counter = Counter + 1
  Loop
  MsgBox("The loop ran " & Counter & " times.")   ' Runs 4 times.
End Sub

Sub CheckWhileLast()
  Dim Counter As Integer = 0
  Dim Number As Integer = 5
  Do
    Number = Number - 1
    Counter = Counter + 1
  Loop While Number > 6
  MsgBox("The loop ran " & Counter & " times.")   ' Runs 1 time.
End Sub
```

تكرار تنفيذ التعليمات إلى أن يتم تحقق الشرط

هناك طريقتان لاستخدام كلمة Until لاختبار شرط فى حلقة Do. يمكن اختبار الشرط قبل الدخول فى حلقة التنفيذ، أو اختباره بعد تنفيذ الحلقة على الأقل مرة واحدة. ويستمر التنفيذ طالما بقى الشرط غير حقيقى (False).

فى المثال التالى، يقوم إجراء CheckUnitlFirst باختبار الشرط قبل الدخول فى الحلقة. فإذا كانت القيمة الابتدائية للمتغير Number تساوى ١٥ بدلا من ٢٠، فإن التعليمات داخل الحلقة لن يتم تنفيذها مطلقا. وفى إجراء CheckUntilLast، يتم تنفيذ التعليمات داخل الحلقة على الأقل مرة واحدة قبل اختبار الشرط، الذى يكون غير حقيقى (False) فى أول اختبار.

```
Sub CheckUntilFirst()
  Dim Counter As Integer = 0
  Dim Number As Integer = 20
  Do Until Number = 15
    Number = Number - 1
    Counter = Counter + 1
  Loop
  MsgBox("The loop ran " & Counter & " times.")   ' Runs 5 times.
End Sub

Sub CheckUntilLast()
  Dim Counter As Integer = 0
  Dim Number As Integer = 20
  Do
    Number = Number - 1
    Counter = Counter + 1
  Loop Until Number = 15
  MsgBox("The loop ran " & Counter & " times.")   ' Runs 5 times.
End Sub
```

الخروج من حلقة Do Loop

ويمكننا الخروج من حلقة Do باستخدام عبارة Exit DO. أحد استخدامات هذه العبارة هو اختبار أحد الشروط التى يمكن أن تتسبب فى حلقة تنفيذ غير منتهية. فإذا كان الشرط حقيقيا، نستخدم عبارة Exit Do للخروج. وعندما يكون الشرط غير حقيقى، يستمر تنفيذ التعليمات بداخل حلقة Do.

فى المثال التالى، يتم تخصيص قيمة للمتغير Number يمكن أن تتسبب فى استمرار تنفيذ حلقة Do أكثر من 2^31 مرة. ولهذا تقوم عبارة If باختبار قيمة المتغير المذكور والخروج من الحلقة عندما تصبح قيمته أقل من صفر.

```
Sub ExitDoExample()
```

```
    Dim Counter As Integer = 0
    Dim Number As Integer = 8
    Do Until Number = 10
      If Number < 0 Then Exit Do
      Number = Number - 1
      Counter = Counter + 1
    Loop
    MsgBox("The loop ran " & Counter & " times.")   ' Runs 8 times.
End Sub
```

حلقات For...Next

تظهر فائدة استخدام حلقات Do عندما لا نعرف مقدما عدد مرات تنفيذ التعليمات داخل الحلقة. وعندما نعرف عدد مرات تنفيذ الحلقة، يصبح من المفضل استخدام حلقة For...Next. وعلى خلاف حلقة Do، تستخدم حلقة For...Next متغيرا يسمى العداد (Counter) تتزايد قيمته أو تتناقص مع كل تكرار للحلقة. والتركيب اللغوى الخاص باستخدام هذه الحلقة، يأخذ الصيغة التالية:

For *counter* = *start* **To** *end* [**Step** *step*]

Next [*counter*]

يجب أن يكون المتغير Counter من نوع البيانات العددية التى تدعم استخدام عامل أكبر من (>)، عامل أقل من (<)، وعامل التساوي (=)، ويكون فى الغالب من نوع الأرقام الصحيحة (Integer). وقيم التكرار التى تتمثل فى start، end، و step هى تعبيرات من نوع الأرقام الصحيحة أيضا. ويمكن أن تكون الفقرة الاختيارية Step موجبة أو سالبة. وعند حذفها، تكون قيمتها الافتراضية مساوية للواحد.

وعندما يبدأ تنفيذ حلقة For...Next، يقوم Visual Basic بتقييم start، end، و step. بعد ذلك يقوم بتخصيص قيمة start للمتغير Counter. وقبل تنفيذ مجمع التعليمات، يقوم بمقارنة المتغير Counter مع قيمة end. إذا تعدت قيمة Counter قيمة end، يتم الخروج من حلقة For وتحويل التحكم إلى العبارة التى تلى عبارة Next. وإلا فإن مجمع التعليمات يتم تنفيذه. وفى كل مرة يقابل فيها Visual Basic عبارة Next، فإنه يقوم بزيادة المتغير Counter بقيمة step ثم يعود إلى عبارة For. مرة أخرى يقوم بمقارنة قيمة Counter مع قيمة end ومرة أخرى إما أن ينفذ مجمع التعليمات أو ينهى تنفيذ الحلقة بناءا على نتيجة

المقارنة. وتستمر هذه العملية إلى أن تتعدى قيمة Counter قيمة end أو تنفيذ عبارة Exit For . ولا يتم إلغاء تنفيذ حلقة التعليمات حتى تتعدى قيمة Counter قيمة end. وتستخدم المقارنة عامل أكبر من (>) عندما تكون قيمة الزيادة موجبة وعامل أقل من (<) عندما تكون قيمة الزيادة سالبة. وعندما تصبح قيمة Counter مساوية لقيمة end، يستمر تنفيذ الحلقة. ويمكن تحديد العداد (Counter) فى عبارة Next لزيادة وضوح البرنامج. ويجب استخدام نفس المتغير المعلن عنة فى عبارة For.

المثال التالى، يقوم بضبط جميع أعضاء مصفوفة على القيمة 128. وتحدد عبارة For متغير العداد k وقيمة البداية وقيمة النهاية. وتقوم عبارة Next بزيادة العداد بمقدار واحد، لأن قيمة step لم يتم تحديدها فى عبارة For.

```
Sub Preset(ByRef A() As Integer)
  Dim I As Integer
  For I = 1 To UBound(A)
    A(I) = 128
  Next I
End Sub
```

ويتم تحديد قيمة البداية والنهاية مرة واحدة فى البداية قبل بدء حلقة For. وإذا تم تغيير هذه القيم داخل تعليمات الحلقة، فإن التغييرات لن تؤثر على تكرار الحلقة. فى الكود السابق، على سبيل المثال، لن يتم استدعاء دالة UBound إلا مرة واحدة عند بدء تنفيذ الحلقة.

زيادة وتخفيض متغير العداد فى حلقة For...Next

باستخدام كلمة Step، يمكننا زيادة أو تخفيض قيمة العداد الخاص بحلقة For بالقيمة التى نحددها. فى المثال التالى، يتم زيادة متغير العداد J بمقدار ٢ فى كل مرة تتكرر فيها الحلقة. وعند نهاية تنفيذ الحلقة، سوف يحتوى المتغير Total على القيم ٢، ٤، ٦، ٨، و ١٠.

```
Sub TwosTotal()
  Dim J, Total As Integer
  For J = 2 To 10 Step 2
    Total = Total + J
    MsgBox("The total is " & Total)
  Next J
```

```
End Sub
```

لتخفيض متغير العداد، نستخدم قيمة سالبة فى Step. وعند القيام بذلك، يجب جعل النهاية أقل من قيمة البداية. فى المثال التالى، يتم تخفيض متغير العداد N بمقدار ٢ فى كل مرة تتكرر فيها الحلقة. وعند انتهاء الحلقة، سوف يحتوى المتغير N على القيم ١٦، ١٤، ١٢، ١٠، ٨، ٦، و ٤. كما أن المتغير Total سوف يحتوى على القيم ١٦، ٣٠، ٤٢، ٥٢، ٦٠، ٦٦، ٧٠.

```
Sub NewTotal()
  Dim N, Total As Integer
  For N = 16 To 4 Step -2
    Total = Total + N
  Next N
  MsgBox("The total is " & Total)
End Sub
```

الخروج من حلقة For...Next قبل وصول العداد إلى القيمة النهائية

يمكن الخروج من حلقة For...Next قبل أن يتخطى العداد قيمة النهاية عن طريق استخدام عبارة Exit For. على سبيل المثال، قد نحتاج إلى الخروج من حلقة لمواجهة أحد الظروف التى تجعل من غير الضروري الاستمرار فى تنفيذ الحلقة. وأيضا، عند اصطياد أحد الإستثناءات بواسطة Try...Catch...Finally، يمكن استخدام عبارة Exit For فى نهاية مجمع تعليمات Finally.

حلقة For Each...Next

تماثل حلقة For Each...Next حلقة For...Next، ولكنها تقوم بتنفيذ مجمع التعليمات لكل عنصر فى مجموعة، بدلا من التنفيذ لعدد محدد من المرات. والتركيب اللغوى لهذه الحلقة، يأخذ الصيغة التالية:

```
For Each elementvariable In collection
  Next [ elementvariable ]
```

ويمكن أن تكون عناصر المجموعة من أى نوع بيانات. ويجب أن يكون نوع بيانات elementvariable يقبل تحويل نوع بيانات أى عنصر من عناصر المجموعة إلية. وفى كل تكرار لهذه الحلقة، يقوم Visual Basic بتخصيص واحد من عناصر المجموعة للمتغير elementvariable وتنفيذ مجمع تعليمات الحلقة. وعندما ينتهى تخصيص جميع عناصر

المجموعة، ينتهي تنفيذ حلقة For Each وينتقل التحكم إلى العبارة التالية لعبارة Next.

الكود التالى يقبل معامل من نوع Form ويضبط لون الخلفية لكل متحكم فى هذا النموذج على اللون الأزرق الفاتح:

```
Sub LightBlueBackground(ByVal ThisForm As System.Windows.Forms.Form)
  Dim ThisControl As System.Windows.Forms.Control
  For Each ThisControl In ThisForm.Controls
    ThisControl.BackColor = System.Drawing.Color.LightBlue
  Next ThisControl
End Sub
```

ونظرا لأن المصفوفات يتم تنفيذها على أنها مجموعات، يمكننا استخدام حلقة For Each...Next للتكرار خلال المصفوفة. الإجراء التالى يقوم بضبط جميع عناصر مصفوفة على القيمة ١٢٨:

```
Sub Preset(ByRef A() As Integer)
  Dim Elt As Integer
  For Each Elt In A
    Elt = 128
  Next Elt
End Sub
```

الخروج من حلقة For Each...Next قبل انتهاء المجموعة

يمكننا الخروج من حلقة For Each...Next قبل انتهاء اختيار كل أعضاء المجموعة باستخدام عبارة Exit For. على سبيل المثال، ربما نريد الخروج من حلقة عند مواجهة أحد الظروف التى تجعل من الضروري الخروج من هذه الحلقة، مثل وجود قيمة غير صحيحة أو استقبال طلب إلغاء. وكذلك عند اصطياد أحد الإستثناءات فى عبارة Try...Catch...Finally، يمكننا استخدام عبارة Exit For فى نهاية مجمع تعليمات Finally.

تعليمات With...End With

فى Visual Basic يجب دائما تحديد أحد الكائنات فى كل عبارة تستدعى واحدا من وسائلة أو تستخدم أحد خصائصه. من ناحية أخرى، إذا كان هناك سلسلة من العبارات التى تعمل على نفس الكائن، يمكن استخدام عبارة With...End With لتحديد الكائن مرة واحدة للاستخدام بواسطة كل العبارات. يؤدى تنفيذ ذلك إلى تشغيل الإجراءات بسرعة

وتجنب تكرار الطباعة. المثال التالى، يقوم بتعبئة نطاق من الخلايا بالعدد ٣٠، يستخدم البنط الأسود العريض، يضبط اللون الداخلى للخلايا على اللون الأصفر.

```
Sub PrepareCells()
  With Worksheets("Sheet1").Range("A1:C10")
    .Value = 30
    .Font.Bold = True
    .Interior.Color = RGB(255, 255, 0)
  End With
End Sub
```

ويمكن جعل عبارات With...End With متداخلة لتحقيق المزيد من الفاعلية. المثال التالى يدرج صيغة فى الخلية A1، ثم يصيغ شكل حروف الطباعة.

```
Sub MyInput()
  With Workbooks("Book1").Worksheets("Sheet1").Cells(1, 1)
    .Formula = "=SQRT(50)"
    With .Font
      .Name = "Arial"
      .Bold = True
      .Size = 8
    End With
  End With
End Sub
```

تعليمات التحكم المتداخلة

يمكن وضع عبارات تعليمات التحكم داخل عبارات تعليمات التحكم الأخرى. على سبيل المثال، يمكن وضع عبارة If...Then...Else داخل حلقة For...Next. والعبارة التى توضع داخل عبارة أخرى يطلق عليها متداخلة (Nested). ويمكن تداخل عبارات التعليمات فى Visual Basic لأى مستوى يتطلبه البرنامج. فى المثال التالى، يقوم الإجراء بإضافة العناصر الموجبة معا بكل صف فى الشبكة:

```
Public Sub SumRows(ByVal A(,) As Double, ByRef R() As Double)
  Dim I, J As Integer
  For I = 0 To UBound(A, 0)
  R(I) = 0
    For J = 0 To UBound(A, 1)
     If A(I, J) > 0 Then
       R(I) = R(I) + A(I, J)
     End If
```

```
        Next J
      Next I
    End Sub
```

تقوم عبارة Next J فى الكود السابق بإقفال حلقة For الداخلية، بينما تقوم عبارة Next I بإقفال حلقة For الخارجية. وبالمثل، تتعلق عبارة End If تلقائيا بأقرب عبارة If لها. وتعمل حلقات Do بنفس الطريقة، حيث ترتبط عبارة Loop بأقرب عبارة Do إليها.

تعليمات Exit

تسمح لنا Exit بالخروج مباشرة من أى هيكل قرارات، حلقة، أو إجراء. وتحول التحكم فى التنفيذ مباشرة إلى العبارة التى تلى أخر عبارة تحكم. ويحدد التركيب اللغوى لعبارة Exit نوع عبارة التحكم التى يتم الخروج منها. وفيما يلى مختلف أشكال صيغ عبارة Exit:

- Exit Select
- Exit Try
- Exit Do
- Exit While
- Exit For

ويمكن أيضا الخروج المباشر من دالة، إجراء، أو خاصية. لتنفيذ ذلك، نستخدم الصيغ التالية:

- Exit Sub
- Exit Function
- Exit Property

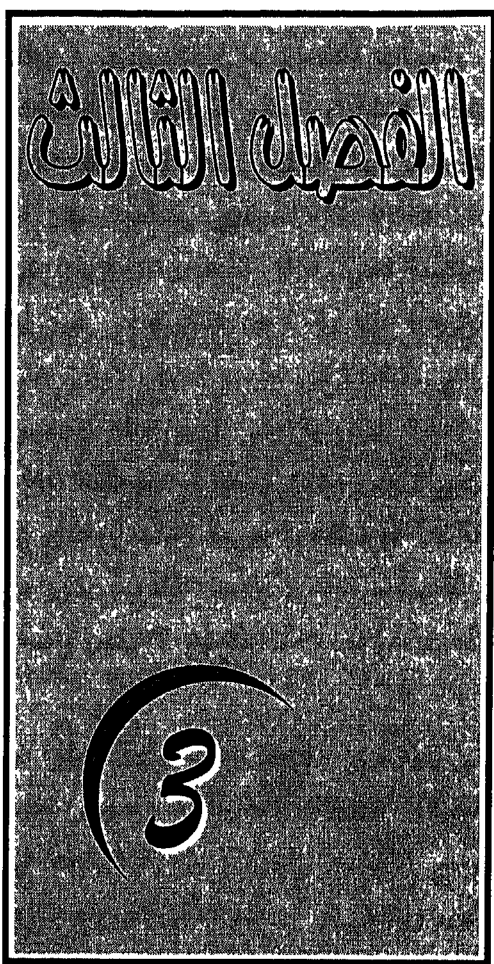
ويمكن استخدام عبارات Exit Sub، Exit Function، وExit Property بأي عدد نحتاج إلية، وفى أى مكان داخل الإجراء، بما فى ذلك عبارات التحكم التى بداخل الإجراء.

الخروج من تعليمات التحكم المتداخلة

عند مواجهة عبارة Exit فى تعليمات تحكم متداخلة، يستمر التنفيذ بدءا من عبارة التعليمات التى تلى نهاية عبارة التحكم الأكثر تداخلا الخاصة بالنوع المحدد فى عبارة Exit. فى المثال التالى، تقع عبارة Exit For فى حلقة For الداخلية، ولهذا ينتقل التحكم فى التنفيذ إلى العبارة التى تلى الحلقة ويستمر فى حلقة For الخارجية.

```
Public Sub InvertElements(ByRef A(,) As Double)
  Dim I, J As Integer
```

```
For I = 0 To UBound(A, 0)
   For J = 0 To UBound(A, 1)
    If A(I, J) = 0 Then
      Exit For     ' Cannot complete this row.
    Else
      A(I, J) = 1 / A(I, J)
    End If
   ' Control comes here if the Exit For is executed.
    Next J
  Next I
End Sub
```



تطبيقات الويندوز

عند تصميم التطبيقات التى تشتمل على واجهات التعامل مع المستخدمين (User Interfaces)، سوف نواجه اثنين من الخيارات: استخدام نماذج الويندوز أو استخدام نماذج الوب. كلا النوعين يتوفر له الدعم الكامل داخل بيئة التطوير المتكاملة (Integrated Development Environment) فى نظام Visual Studio أثناء التصميم. وكلاهما يمكن أن يوفر واجهة استخدام غنية ووظائف تطبيقية متقدمة لحل مشاكل الأعمال التجارية. على هذا الأساس، تواجهنا مشكلة اختيار التكنولوجيا المناسبة لأحد التطبيقات. غير أن خصائص التطبيقات التى نقوم بتكوينها، هى التى تساعدنا فى تحديد الخيار الأمثل. على سبيل المثال، عند القيام بإنشاء تطبيق موقع تجارة إلكترونية على الوب يمكن للجمهور الوصول إليها عبر شبكة الإنترنت، من الطبيعى تكوين تطبيق يستخدم صفحات نماذج الوب. وإذا كنا بصدد إنشاء تطبيق كثير الإجراءات، ذو استجابة سريعة وعالية ويحتاج إلى الاستفادة من الوظائف الكاملة للكمبيوتر، مثل التطبيقات الخاصة بأعمال المكتب، يكون من الطبيعى استخدام نماذج الويندوز. من ناحية أخرى، نجد أن هذا الخيار لا يكون واضحا بما فيه الكفاية فى بعض الحالات الأخرى.

تعتبر تطبيقات الويندوز من التطبيقات التى يقوم فيها التطبيق العميل بكمية كبيرة من المعالجة، مثل تطبيقات سطح الكمبيوتر التقليدية، تطبيقات الرسومات والصور، أنظمة إدخال البيانات، أنظمة نقاط البيع، و برامج الألعاب. وتشترك هذه التطبيقات فى اعتمادها على قوة الكمبيوتر فى المعالجة والعرض. وبالنظر إلى أن هذه التطبيقات يتم بناؤها داخل إطار نظام تشغيل الويندوز، لذا يتوفر لها إمكانية الوصول إلى موارد النظام بالكمبيوتر، مثل الملفات المحلية، سجل الويندوز، الطابعة، وغيرها من الموارد.

ويتم بناء تطبيقات الويندوز فىVisual Studio باستخدام نظام NET Framework، الذى يتكون من مجموعة غنية من التصنيفات التى تسمح لنا ببرمجة التطبيقات المتطورة. وتتيح لنا تطبيقات الويندوز التى يجرى إنشاؤها باستخدام تصنيفات NET مزايا كثيرة، تتمثل فى إمكانية الوصول إلى خدمات نظام التشغيل والاستفادة من المزايا المتوفرة فى بيئة الكمبيوتر الخاصة بالمستخدم، الوصول إلى البيانات باستخدام ADO NET، تنفيذ الرسومات المتقدمة باستخدام تصنيفات +GDI. كما يمكن أن تقوم تطبيقات الويندوز باستدعاء وسائل متاحة من خلال خدمات الوب فى صورة XML، مما يشجع على الاستفادة من معلومات

وموارد الكمبيوتر الواردة من مصادر متعددة.

وكما هو الحال فى تطبيقات نظام NET الأخرى، يمكن إنشاء تطبيقات الويندوز فى محرر نصوص، استدعاء وسائل NET وتصنيفاته، ترجمة البرنامج باستخدام سطر الأوامر، وتوزيع البرنامج التنفيذى الناتج. كما يمكن أيضا استخدام نظام Visual Studio .NET فى بناء تطبيقات الويندوز، مما يؤدى إلى توفير الكثير من الوسائل المتاحة فى ذلك النظام، التى تجعل بناء التطبيقات أسرع كثيرا وأكثر موثوقية. تشمل هذه الوسائل ما يلى :

- أدوات التصميم المرئية لنماذج الويندوز.
- محررات الكود الذكية التى تستطيع استكمال العبارات، مراجعة القواعد اللغوية، وغيرها من الأدوات.
- أدوات ترجمة البرامج والبحث عن الأخطاء.
- تسهيلات إدارة المشروع لإنشاء وإدارة ملفات التطبيقات، بما فى ذلك نشر التطبيقات محليا، أو على شبكة الإنترنت وشبكة الإنترنت.

نماذج الويندوز

بالنظر إلى أن النموذج هو الوحدة الأساسية التى تتكون منها التطبيقات، يصبح من الضروري الاهتمام بتصميماتها ووظائفها. من الناحية الجوهرية، يمكن تعريف النموذج بأنة عبارة عن لوح يقوم المبرمج بتطويره عن طريق إضافة أدوات التحكم المختلفة إلية بهدف تشكيل واجهة التعامل مع المستخدم، وإضافة الكود إلية للتعامل مع البيانات. وتعتبر نماذج الويندوز هى ساحة العمل الجديدة المستخدمة لتطوير تطبيقات مايكروسوفت ويندوز داخل نظامNET Framework الجديد الذى قدمته مايكروسوفت. وهو نظام يوفر لنا مجموعة قوية من تصنيفات برمجة الكائنات التى تمكننا من تطوير تطبيقات نماذج ويندوز متقدمة.

ويمكن استخدام النموذج لعرض المعلومات أمام المستخدم وقبول الإدخالات منة. ويمكن أن يحتوى التطبيق على نافذة وحيدة (SDI)، نوافذ لوثائق متعددة (MDI)، مربعات حوار، أو واجهة عرض للرسومات. وأسهل طريقة لتحديد واجهة بينية مع المستخدم بالنسبة لنموذج هى وضع أدوات تحكم على وجه هذا النموذج. وتعتبر النماذج كائنات تتيح استخدام خصائص تتحكم فى طريقة عرض هذه النماذج، وسائل تتحكم فى سلوكها،

وأحداث تحدد التفاعل مع المستخدم. ويتم إعداد كائن النموذج لمقابلة متطلبات التطبيقات عن طريق ضبط خصائصه وكتابة الكود اللازم للاستجابة إلى أحداثه.

وكما هو الحال بالنسبة لجميع الكائنات فى نظام NET Framework، نجد أن النموذج يعتبر مثل من تصنيف نقوم بإنشائه باستخدام مصمم النماذج (Windows Forms Designer). وعندما نعرض مثل من النموذج فى وقت التشغيل، فإن تصنيف النموذج (Form Class) يعتبر هو القالب الذى نستخدمه فى بناء ذلك النموذج. ويسمح لنا NET Framework بالوراثة من النماذج الموجودة بغرض إضافة المزيد من الوظائف أو تعديل السلوك القائم. وعندما نضيف نموذج إلى مشروع، يمكن وراثته من تصنيف Form الأساسى الموجود فى NET Framework أو من نموذج سبق تكوينه.

تكوين نماذج الويندوز

الحل السريع لتكوين وتعديل نماذج الويندوز هو استخدام مصمم النماذج (Windows Designer)، مع أن النموذج يمكن تكوينه بالكامل فى محرر الكود (Code Editor). وتشتمل عملية التكوين على عدد من الخطوات، أهمها: ضبط خصائص النموذج (Setting Form Properties)، وضع أدوات التحكم (Controls) على وجه النموذج. وقبل تكوين النموذج، يجب أن يكون هناك مشروع تطبيق يستخدم هذا النموذج.

مشروعات نماذج الويندوز

فى داخل مشروع نماذج الويندوز، يعتبر النموذج هو الأداة الأساسية المستخدمة للتفاعل مع المستخدم. ويمكننا عن طريق ضم مجموعات مختلفة من أدوات التحكم وكتابة الكود اللازم، الحصول على المعلومات من المستخدم والاستجابة لطلباته، العمل مع مخازن البيانات الموجودة، و استعلام نظام الملفات ثم الكتابة به. ويمكن أن يكون مشروع نماذج الويندوز فى صورة ملف تنفيذي قائم بذاته أو يمثل طبقة العميل فى التطبيقات التى تتكون من طبقتين (Two-tier)، طبقة الخادم وطبقة العميل.

تكوين مشروعات تطبيقات الويندوز

لتكوين مشروع تطبيق ويندوز، نقوم بتنفيذ الخطوات التالية:

- فى قائمة File، نشير إلى New، ثم نختار Project. يؤدى ذلك إلى ظهور مربع حوار New Project.

- فى الجانب الأيسر من مربع حوار New Project، نختار Visual Basic. وفى الجانب الأيمن، نختار Windows Application.
- نقوم بإدخال اسم للمشروع فى مربع Name، وفى مربع Location، ندخل المسار الذى يتم فيه حفظ المشروع
- ننقر OK لفتح مصمم نماذج الويندوز (Windows Forms Designer) وبه النموذج الرئيسى بالتطبيق.

مخططات نماذج الويندوز

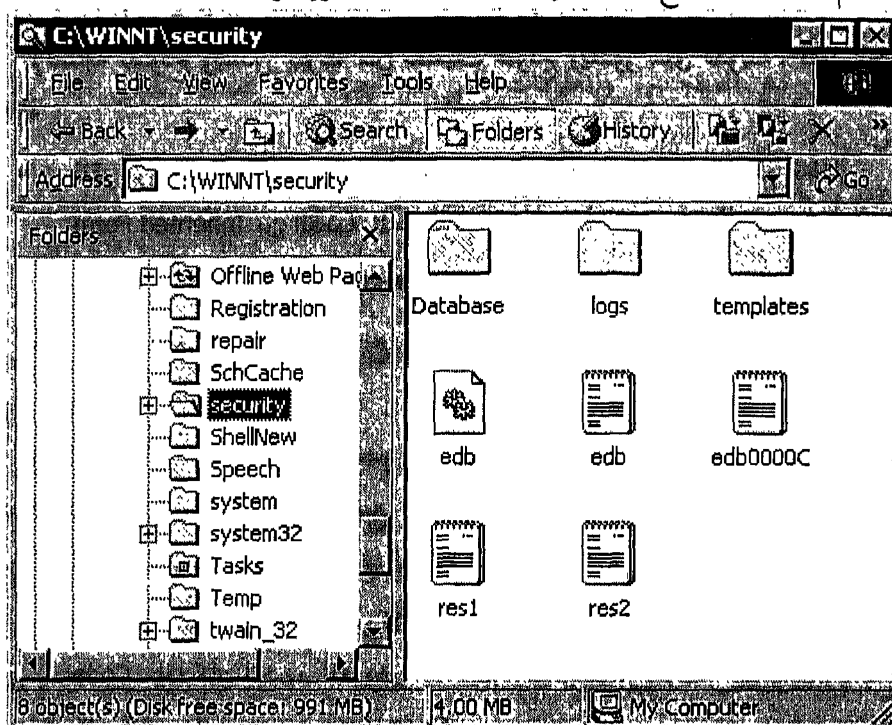
هناك ثلاثة أنواع من الواجهات البينية (Interfaces) للتعامل مع المستخدمين :

- واجهة التعامل مع وثيقة واحدة (the single-document interface) SDI.
- واجهة التعامل مع وثائق متعددة (the multiple-document interface) MDI.
- واجهة التعامل مع مستكشف (the Explorer-style interface).

من أمثلة واجهة التعامل مع وثيقة واحدة، تطبيق WordPad الموجود فى مايكروسوفت ويندوز. فى هذا التطبيق، يمكن فتح وثيقة واحدة ويجب إغلاقها قبل فتح وثيقة جديدة. .ومن أمثلة واجهات التعامل مع وثائق متعددة، Microsoft Excel الذى يسمح لنا بعرض عدة وثائق فى نفس الوقت، كل وثيقة فى نافذة خاصة بها. ويمكن التعرف على التطبيقات ذات الواجهات متعددة الوثائق عن طريق وجود قائمة ويندوز تحتوى على أوامر للتنقل بين الوثائق المفتوحة المختلفة. ولتحديد نوع الواجهة المفضل استخدامها، نحتاج إلى تحديد الغرض من التطبيق. على السبيل المثال، التطبيق الذى يقوم بمعالجة مطالبات التأمين يمكن أن يحتاج إلى استخدام الوجهة متعددة الوثائق، لكى يستطيع الموظف العمل مع أكثر من وثيقة والتنقل بينها فى نفس الوقت. بينما تكون الواجهة ذات الوثيقة الواحدة أكثر ملاءمة لتطبيقات التقاويم لأننا لا نحتاج إلى عرض أكثر من تقويم فى نفس الوقت. وتعتبر واجهة الاستخدام ذات الوثيقة الواحدة هى المخطط الأكثر شيوعا بالنسبة لتطبيقات الويندوز.

بالإضافة إلى هذين النوعين الشائعين من واجهات الاستخدام، هناك واجهه بينية أخرى أخذة فى الانتشار هى واجهة المستكشف (Explorer interface)، التى يوضحها

الشكل رقم (٨). تتكون هذه الواجهة من نافذة واحدة تحتوى على جانبين أو قسمين، أحد القسمين يحتوى على شجرة معلومات فى يسار النافذة، والقسم الثانى عبارة عن منطقة عرض فى الجانب الأيمن، كما فى مايكروسوفت اكسبلورر. ويناسب هذا النوع من واجهات الاستخدام، عمليات تصفح عدد كبير من المستندات، الصور، والملفات.



شكل رقم ٨

إضافة نماذج الويندوز إلى مشروع

يمكن أن يحتاج تطبيق الويندوز الذى نقوم بتكوينه، إلى نماذج أخرى غير النموذج الإفتراضى. نظام NET Framework يجعل من السهل القيام بهذه المهمة عن طريق إتاحة أنواع مختلفة من النماذج، مثل مربعات حوار (Dialog Boxes)، شاشات بدء التشغيل (Startup Screens)، وغير ذلك من أنواع نماذج الدعم.

لإضافة نموذج ويندوز يرث من تصنيف Form، ننفذ الخطوات التالية:

- نقر بزر الماوس الأيمن على المشروع فى مربع Solution Explorer، نختار Add ثم

Add Windows Form من القائمة المختصرة.

- فى مربع حوار Add New Item، نختار Windows Form فى جانب Templates، ندخل اسم النموذج فى مربع Name ثم ننقر Open لعرض النموذج الجديد فى مصمم النماذج.

لإضافة نموذج ويندوز يرث من تصنيف نموذج ويندوز سبق تكوينه ، ننفذ الخطوات التالية :

- ننقر بزر الماوس الأيمن على المشروع فى Solution Explorer، ثم نختار Add، Inherited Form من القائمة المختصرة.
- فى مربع حوار Add New Item، نختار Inherited Form فى جانب Templates، ندخل اسم النموذج فى مربع Name ثم ننقر Open لعرض النموذج الجديد فى مصمم النماذج.

ويمكن عرض النموذج فى مصمم النماذج (Forms Designer) باستخدام مشهدين، المشهد الأول يعرض النموذج فى وضع التصميم (Designer View)، والمشهد الثانى يعرض الكود الخاص بالنموذج (Code View). لعرض النموذج فى أحد المشهدين، ننفذ ما يلى:

- ننقر على النموذج بزر الماوس الأيمن فى مربع Solution Explorer، ثم نختار View Code، إذا كان المشهد المعروض هو Designer View. ونختار View Designer، إذا كان المشهد العروض هو Code View.

اختيار نموذج بدء التشغيل فى تطبيقات الويندوز

عندما نضيف نموذج جديد إلى مشروع، لن يقوم هذا النموذج بعرض نفسة تلقائيا فى وقت التشغيل. من ناحية أخرى، سوف يكون النموذج الذى يتم تكوينه عندما نختار Windows Application فى مربع حوار New Project، هو نموذج بدء التشغيل الإفتراضى. لتغيير نموذج بدء التشغيل Project Property Page، ننفذ الخطوات التالية:

- ننقر بزر الماوس الأيمن على المشروع فى مربع Solution Explorer ثم نختار Properties من القائمة المختصرة. يترتب على ذلك فتح صفحة خصائص المشروع.
- نختار نموذج بدء التشغيل من القائمة المنسدلة فى مربع Startup Object.

ضبط الخصائص بنموذج الويندوز

الخصائص هى الصفات التى تعرف الحالة، السلوك، والمظهر لأحد النماذج أو أدوات التحكم. وتعرض معظم أدوات التحكم خصائص لاستخدامها فى تعريف العناصر التى تكون شكل ظهورها، مثل ألوان الحروف، لون الخلفية، الحجم، الموقع، و حروف الطباعة المستخدمة لعرض النصوص. ويمكن للنماذج وأدوات التحكم أن تعرض أيضا خصائص عن كيفية التعامل مع المستخدم، والمعلومات التى تحتاجها أداة التحكم للعمل أثناء وقت التشغيل.

وتنقسم خصائص النموذج أو أداة التحكم إلى قسمين. القسم الأول يحتوى على الخصائص التى يمكن مشاهدتها وضبطها أثناء التصميم من خلال نافذة الخصائص (Properties Window). والقسم الثانى يحتوى على الخصائص التى لايمكن ضبطها إلا باستخدام الكود. وفيما يلى نعرض أهم خصائص النموذج الذى يجب ضبطها باستخدام نافذة الخصائص أو باستخدام الكود.

جعل النموذج غير مرئى عند بدء التشغيل

يتم التحكم فى ظهور النموذج بضبط خاصية Visible. ومن الملاحظ أن ضبط هذه الخاصية بالنسبة للنموذج الرئيسى على القيمة False، لن يكون له تأثير وسوف يستمر النموذج فى الظهور. يرجع السبب فى ذلك إلى أن حياة التطبيق ترتبط بفترة بقاء النموذج الرئيسى. ولذلك يمكن التغلب على هذه المشكلة بفصل فترة بقاء التطبيق عن فترة بقاء النموذج بوضع كود بدء التشغيل فى وحدة منفصلة عن وحدة النموذج. بمجرد تنفيذ ذلك، يصبح فى الإمكان إخفاء النموذج الرئيسي أو إظهاره لأن التطبيق لن ينتهى إلا بإغلاق وحدة الكود المذكورة.

لضبط نموذج لكى يكون غير ظاهر عند تشغيله، نقوم بتنفيذ الخطوات التالية :

١. نضيف وحدة كود إلى تطبيق ويندوز عن طريق النقر بزر الماوس الأيمن على المشروع واختيار Add Module من القائمة المختصرة.

٢. نختار Module فى جانب Templates بمربع حوار Add New Item، نحدد اسم للوحدة ثم ننقر Open لعرض مربع وحدة الكود.

٣. داخل وحدة الكود التى تم إضافتها، نضيف برنامج فرعى (Subroutine) باسم

Main لإستخدامة فى بدء تشغيل للمشروع ثم نضيف الكود التالى إلية :

```
Sub Main ()
    Dim f1 as New Form1 ()
    System.Windows.Forms.MessageBox.Show ( _
     "The application is running now, but no forms have been shown.")
    f1.Text = "Running Form"
    f1.ShowDialog ()
End Sub
```

٤. تغيير كائن بدء البرنامج بالنسبة للمشروع إلى Sub Main بدلا من Form1.

٥. نضغط F5 لتشغيل المشروع .

عندما يتم تشغيل التطبيق السابق، يجرى تنفيذ الكود الموجود داخل البرنامج الفرعى Main() أولا و يختفى النموذج Form1 إلى أن يتم تشغيل الكود الذى يقوم بعرضه. يمكننا ذلك من فعل ما نريده داخل النموذج فى خلفية التطبيق بدون علم المستخدم.

وعند عرض أحد النماذج، يمكننا التحكم فى وضعها على القمة أمام جميع النماذج الأخرى باستخدام خاصية TopMost. ويترتب على وضع نموذج على القمة، بقاء هذا النموذج طافيا أمام جميع النماذج الأخرى فى التطبيق أو فى جميع التطبيقات بناءا على نظام التشغيل المستخدم. فى نظام تشغيل مايكروسوفت ويندوز 2000، يبقى نموذج القمة أمام جميع النوافذ فى نفس التطبيق. وفى نظام تشغيل مايكروسوفت 98، يبقى نموذج القمة أمام جميع نماذج التطبيقات العاملة.

لوضع نموذج على القمة فى تطبيق نماذج ويندوز فى وقت التصميم :

- نختار النموذج فى نافذة التصميم (Designer View).
- فى نافذة الخصائص، نجعل خاصية TopMost تساوى True.

لوضع نموذج على القمة وقت التشغيل، نجعل خاصية TopMost تساوى True باستخدام الكود التالى :

```
Public sub MakeOnTop ()
  myTopForm.TopMost = True
End Sub
```

جعل نماذج الويندوز شفافة

يمكننا التحكم فى شفافية النوافذ التى يجرى عرضها باستخدام خاصية Opacity. ويجب ملاحظة أن النماذج الشفافة مدعمة فقط فى نظام التشغيل ويندوز 2000 والإصدارات التى بعدة. لتغيير درجة شفافية الويندوز وقت التصميم، نجعل قيمة خاصية Opacity بين القيمة 0.0 وبين القيمة 1.0. تمثل القيمة 0.0 الشفافية الكاملة، بينما تمثل القيمة 1.0 الإعتام الكامل. وللتحكم فى هذه الشفافية وقت التشغيل، ندخل الكود التالى فى إجراء معالجة أحد الأحداث:

```
Public Sub MakeSeeThru ()
  frmTransparentForm.Opacity = 0.83
End Sub
```

عرض النماذج باستخدام نمط Modal أو نمط Modeless

تكون النماذج ومربعات الحوار إما Modal أو Modeless. النموذج من نوع Modal أو مربع الحوار (Dialog Box) يجب إغلاقه قبل الاستمرار مع باقى أجزاء التطبيق. مربعات الحوار التى تعرض رسائل مهمة يجب دائما أن تكون Modal. من أمثلة مربعات الحوار من نوع Modal، مربع حوار About فى Visual Studio. كما أن مربع MessageBox يعتبر من نماذج Modal التى يمكن استخدامها.

وتتميز النماذج من نوع Modeless بأنها تسمح بتحويل بؤرة التركيز بين نموذج وأخر بدون الحاجة إلى إغلاق النموذج الأساسى. وتعتبر النماذج من نوع Modeless أصعب فى البرمجة لأن المستخدم يمكنة الوصول إليها بترتيب غير متوقع. ويجب على المبرمج أن يحافظ فى نفس الوقت على ثبات حالة التطبيق بغض النظر عن فعل المستخدم. وفى الغالب يتم عرض أدوات الويندوز باستخدام هذا الأسلوب. ومن أمثلة مربعات الحوار من نوع Modeless، مربع حوار Find الذى يمكن الوصول إلية من قائمة Edit فى Visual Studio. ومن المفضل استخدام النماذج من هذا النوع لعرض الأوامر أو المعلومات المتكررة الاستخدام.

لعرض النموذج فى صورة مربع حوار من نوع Modal، نستخدم وسيلة ShowDialog. الكود التالى يوضح كيفية استخدام هذه الوسيلة:

```
Dim frmAbout as New Form ()
frmAbout.ShowDialog ()
```

تحتوى هذه الوسيلة على معامل owner، الذى يمكن استخدامه فى تحديد العلاقة بين الأصل والتابع بالنسبة لنموذج. على سبيل المثال، عندما يقوم نموذج أساسى بعرض مربع حوار، يمكن تمرير كلمة Me إلى مربع الحوار لتحديد أن النموذج الأساسى هو المالك. يوضح الكود التالى استخدام هذا المعامل :

```
Private Sub mnuAbout_Click (ByVal sender As Object, ByVal e As System.EventArgs) Handles mnuAbout.Click
    Dim f As New Form ()
    f.ShowDialog (Me)
End Sub
```

ولعرض نموذج فى صورة مربع حوار من نوع Modeless، نستخدم وسيلة Show. يوضح الكود التالى كيفية عرض مربع حوار About باستخدام هذه الوسيلة :

```
Dim f As New Form ()
f.Show ()
```

ويجب ملاحظة أن عرض نموذج فى صيغة Modal، يترتب علية عدم تنفيذ الكود التالى لوسيلة ShowDialog إلى أن يتم إغلاق مربع الحوار. وعند عرض نموذج فى صيغة Modeless، يتم تنفيذ الكود الذى يلى وسيلة Show مباشرة بعد عرض النموذج.

تغيير حدود نماذج الويندوز

هناك العديد من أنماط الحدود التى يمكن الاختيار من بينها عند تحديد شكل نماذج الويندوز التى نقوم بتكوينها. عن طريق تغيير خاصية FormBorderStyle، يمكننا التحكم فى سلوك تغيير حجم النموذج. بالإضافة إلى ذلك، يؤثر ضبط خاصية FormBorderStyle على كيفية عرض شريط العنوان وعلى الأزرار التى يمكن أن تظهر به. الجدول رقم (١٦) يعرض أنماط الحدود التى يمكن استخدامها مع النماذج.

الضبط	الإيضاح
None	لا يوجد نمط معين من الحدود. ويستخدم فى نماذج بدء التشغيل.
Fixed 3D	يجعل حدود النموذج ذات ثلاثة أبعاد ولا يسمح بتغيير حجم النموذج. يمكن أن يشتمل على مربع قائمة تحكم، عنوان،

الضبط	الإيضاح
	أزرار تصغير وتكبير. يكون حدود مرتفعة بالنسبة لجسم النموذج.
Fixed Dialog	يستخدم مع مربعات الحوار ولا يسمح بتغيير حجم النموذج. يمكن أن يشتمل على مربع قائمة التحكم، عنوان، أزرار تصغير وتكبير. يجعل الحدود منخفضة.
Fixed Single	لا يسمح بتغيير حجم النموذج. يمكن أن يشتمل على مربع قائمة تحكم، شريط عنوان، أزرار تصغير وتكبير. يمكن تغيير حجمة فقط باستخدام أزرار التصغير والتكبير. يجعل الحدود تتكون من خط منفرد.
Fixed Tool Window	يستخدم مع نوافذ الأدوات. يعرض نافذة لا تقبل تغيير الحجم، بها زر إغلاق ونص عنوان. لا يظهر النموذج فى شريط المهام.
Sizable	يمثل الضبط الإفتراضى ويستخدم فى الغالب مع النافذة الرئيسية. يسمح بتغيير الحجم. يمكن أن يشتمل على مربع قائمة تحكم، شريط عنوان، أزرار تصغير وتكبير. يمكن تغيير الحجم باستخدام مربع التحكم، أزرار التصغير والتكبير، أو الماوس.
Sizable Tool Window	يستخدم مع نوافذ الأدوات. يعرض نافذة قابلة لتغيير الحجم، وتحتوى على زر الإغلاق وعنوان. ولا يظهر هذا النموذج فى

الضبط	الإيضاح
	شريط المهام.

جدول ١٦

لضبط نمط حدود نماذج الويندوز فى وقت التصميم :

- فى نافذة Properties، نضبط خاصية FormBorderStyle على النمط المطلوب. وسوف يتحكم نمط الحدود الذى يتم اختياره فى عرض مربعات Minimize و Maximizeعلى شريط العنوان .

ويتم ضبط نمط حدود النموذج باستخدام تعداد FormBorderStyle. وبالتالى يمكن ضبط أو تغيير نمط حدود النموذج باستخدام الكود عن طريق ضبط خاصية FormBorderStyle على إحدى القيم الموجودة فى ذلك التعداد. الكود التالى يبين كيفية ضبط خاصية نمط حدود نموذج DlgBx1 على قيمة FixedDialog :

```
DlgBx1.FormBorderStyle=
ystem.Windows.Forms.FormBorderStyle.FixedDialog
```

بالإضافة إلى ذلك، عند اختيار نمط حدود يسمح للنموذج بامتلاك أزرار Minimize و Maximize، يمكن تحديد ما إذا كنا نرغب فى تشغيل واحد منهما أو كلاهما. من الناحية الافتراضية، يتم إتاحة أزرار Minimize وMaximize تلقائيا، كما يتم التعامل مع وظائفها من خلال نافذة Properties. ولحجب أزرار Minimize وMaximize فى نماذج ويندوز :

- فى نافذة Properties،نختار إما خاصية Form.MinimizeBox أو Form.MaximizeBoxثم نختار القيمة False. وبناءا على الخاصية التى نقوم بضبطها، سوف يتم عرض الزر المقابل.

تغيير أحجام نماذج الويندوز

يمكن تغيير حجم النموذج بدويا باستخدام نافذة تصميم نماذج الويندوز (Windows Forms Designer)، من خلال نافذة الخصائص (Properties Window)، أو باستخدام الكود. لتغيير حجم النموذج باستخدام نافذة تصميم النماذج فى وقت التصميم:

- فى نافذة مصمم نماذج الويندوز، ننقر على النموذج لاختياره.
- ننقر ونسحب واحدا من الثمانية مقابض التى تظهر على حدود النموذج. تشبه مقابض التحكم فى الحجم مربعات صغيرة بيضاء، يتحول مؤشر الماوس عندها إلى سهم ذو رأسين عند الإشارة إلى المقبض. ويمكن التحكم فى حجم النموذج أيضا عن طريق ضغط مفتاح AROW أثناء الاحتفاظ بمفتاح SHIFT للأسفل.

لتغيير حجم نموذج باستخدام نافذة Properties فى وقت التصميم:

- فى نافذة Properties، ننقر على خاصية Size وندخل قيم خاصة بالعرض والارتفاع، مع وضع فاصلة بينهما. ويمكن توسيع هذه الخاصية للفصل بين إدخال قيم العرض (Width) وقيم الارتفاع (Height). ولتغيير حجم نموذج باستخدام الكود، نضبط خاصية Size، كما يتضح من الكود التالى:

```
Form1.Size = New System.Drawing.Size (100, 100)
```

ويمكن تغيير قيمة خاصية Width وخاصية Height، كل على انفراد. فى الكود التالى، يتم ضبط خاصية Width على قيمة 300 بكسل بدءا من الحافة اليسرى للنموذج، بينما تبقى خاصية Height ثابتة:

```
Form1.Width = 300
```

كما يمكن تغيير حجم نموذج عن طريق زيادة قيم خصائص Width أو Height، كما يتضح من الكود التالى :

```
Form1.Width += 200
```

ضبط مواقع نماذج الويندوز

يمكن تحديد مكان عرض النموذج على شاشة الكمبيوتر عن طريق ضبط خاصية Location، التى تحدد موقع الركن الأيسر العلوي للنموذج. كما نحتاج أيضا إلى ضبط خاصية StartPosition للإشارة إلى حدود منطقة العرض.

لتحديد موقع النموذج باستخدام نافذة الخصائص :

- فى نافذة Properties، نختار النموذج من القائمة المنسدلة ثم نضبط خاصية StartPosition على Manual.
- ندخل قيم خاصية Location وفصلها باستخدام الفاصلة، حيث يمثل الرقم الأول

(X) المسافة اعتبارا من الحد الأيسر لمنطقة العرض والرقم الثانى (Y) يمثل المسافة من الحد الأعلى لهذه المنطقة.

لتحديد موقع النموذج باستخدام الكود :

- نحدد موقع النموذج وقت التشغيل عن طريق ضبط خاصية Location، كما هو موضح بالكود التالى :

```
Form1.Location = New Point (100, 100)
```

- أو نغير الإحداثي X أو الإحداثي Y لموقع النموذج عن طريق استخدام الخاصية الفرعية Left فى مقابل الإحداثي X والخاصية الفرعية Top فى مقابلة الإحداثي Y ، كما يتضح من الكود التالى :

```
Form1.Left = 300
```

ويمكن تغيير موقع النموذج عن طريق زيادة قيمة الإحداثي X باستخدام الخاصية الفرعية ، Left، كما يتضح من الكود التالى :

```
Form1.Left += 200
```

ومن الملاحظ أن ضبط خاصية Location يترتب علية ضبط المواقع X وY معا. ولضبط الموقعين كل على انفراد، نستخدم الخاصية الفرعية Left(X) والخاصية الفرعية Top(Y). وبدلا من استخدام خاصية Location، يمكن استخدام خاصية DesktopLocation لضبط موقع النموذج. تقوم هذه الخاصية بضبط موقع النموذج بالنسبة إلى شريط المهام (Taskbar). وتعتبر مفيدة عند تثبيت شريط المهام فى قمة أو على يسار وحدة العرض. تثبيت شريط المهام بهذه الطريقة يحجب إحداثيات سطح الكمبيوتر (0,0). والنموذج الذى يتم ضبط خاصية DesktopLocation به على القيم (0,0)، دائما يظهر فى الركن الأيسر الأعلى من وحدة العرض، ولكن ليس خلف شريط المهام.

لضبط خاصية DesktopLocation، نضبط الخاصية على موقع جديد للنموذج، كما يتضح من الكود التالى:

```
Dim frmAccounts as new Form ()
FrmAccounts.DesktopLocation = new Point (100,100)
```

ومن الملاحظ أن خاصية DesktopLocation لا تظهر فى نافذة الخصائص ويمكن ضبطها فقط من خلال الكود.

الوراثة فى نماذج الويندوز

تظهر أهمية وراثة النماذج عندما يحتاج المشروع إلى نموذج سبق تكوينه فى مشروع سابق. كما يمكن تصميم نموذج يحتوى على أدوات التحكم التى نحتاج إليها ثم جعلة قالبا للوراثة. يمكن بعد ذلك وراثة ذلك القالب وإدخال بعض التعديلات التى يحتاج إليها التطبيق على النسخ الموروثة مع الاحتفاظ بالخصائص الموجودة فى النموذج الأصلى. وتعتبر الوراثة وسيلة سهلة لنسخ أفضل المجهودات بدون الحاجة إلى بدء عملية تكوين النموذج من البداية فى كل مرة نحتاج فيها إلى النموذج. وللوراثة من نموذج، يجب بناء الملف أو منطقة الأسماء (Namespace) التى تحتوى على النموذج فى صورة ملف exe أو ملف dll، كما يجب إضافة مرجع خاص بمنطقة الأسماء إلى التصنيف الذى يقوم بوراثة النموذج.

وراثة نماذج الويندوز باستخدام الكود

للوراثة من نموذج باستخدام الكود، ننفذ الخطوات التالية :

١. نضيف مرجع إلى منطقة الأسماء التى تحتوى على النموذج المطلوب وراثته، فى التصنيف.

٢. نضيف مرجع خاص بالنموذج المطلوب وراثته إلى تعريف التصنيف. ويجب أن يشتمل المرجع على منطقة الأسماء التى يوجد بها النموذج واسم النموذج المطلوب وراثته، كما يتضح من الكود التالى:

```
Public Class Form2 : Inherits Namespace1.Form1
```

ومن الممكن أيضا، أن تكون صيغة المرجع كما يلى:

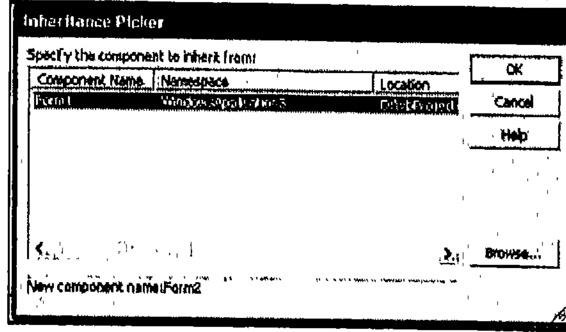
```
Public Class Form2
Inherits Namespace1.Form1
```

وراثة النماذج باستخدام مربع حوار Inheritance Picker

يعتبر استخدام مربع حوار Inheritance Picker أسهل طريقة لوراثة نموذج أو كائن أخر. يمكننا استخدام هذه الطريقة من الاستفادة من الكود والواجهات البينية (Interfaces) التى سبق تكوينها فى الحلول الأخرى. وقبل وراثة أحد النماذج باستخدام هذه الطريقة، يجب أولا بناء النموذج المطلوب وراثته فى صورة ملف exe أو ملف dll. لاستخدام مربع حوار Inheritance Picker فى وراثة نموذج، نتبع الخطوات التالية:

١. من قائمة Project بالقائمة الرئيسية، نختار Add New Item.

٢. فـى مـربع حوار Add New Item، نختار Local Project Items من الجانب الأيمن ومـن الجانب الأيسر نختار Inherited Form. ندخل اسم للنموذج فى مربع Name ثـم نـنقر Open. يترتـب عـلى ذلـك فـتح مـربع حوار Inheritance Picker، المبين بالشكل رقم (١٠).



شكل رقم ١٠

٣. داخـل مـربع حـوار Inheritance Picker، نـنقر زر Browse. نخـتار الملـف الـذى يحـتوى عـلى النموذج المطلوب وراثته ثم ننقر Open للعودة إلى مربع Inheritance Picker.

٤. نختار المكون (Component) فى مربع الحوار المذكور ثم ننقر Ok. يترتب على ذلك إضافة المكون إلى المشروع .

تأثير تعديل شكل ظهور نموذج الأساس

أثناء تطوير التطبيق، قد نحتاج إلى تغيير الشكل الذى يظهر به نموذج الأساس الذى يتم الوراثة منة. تنعكس التغييرات التى تحدث لشكل نموذج الأساس على النماذج الموروثة منة أثناء تصميم التطبيقات، عند بناء المشروع الذى يحتوى على نموذج الأساس. وليس كافيا حفظ التغييرات التى تحدث فى نموذج الأساس لكى تنعكس على النماذج الموروثة.

الأحداث فى نماذج الويندوز

الحدث هو شئ ما يحدث لأحد الكائنات، وفى الغالب يكون نتيجة لتفاعل المستخدم مع الكائن. على سبيل المثال، يوجد بكائن النموذج حدث النقر (Click) الذى يحدث عند قيام المستخدم بالنقر على النموذج باستخدام الماوس. وكل نموذج أو متحكم يعرض مجموعة سابقة التعريف من الأحداث التى يمكن برمجتها بتخصيص إجراءات معالجة لها. تحتوى

إجراءات المعالجة على تعليمات يتم تنفيذها عند وقوع أحد هذه الأحداث. وتقوم التطبيقات التى تقودها الأحداث باستدعاء هذه الإجراءات استجابة لوقوع الحدث.

مصادر الاحداث فى نماذج الويندوز

تختلف أنواع الأحداث المرتبطة بالكائنات المختلفة، ولكن هناك أنواع كثيرة مشتركة بين معظم هذه الكائنات. على سبيل المثال، معظم أدوات التحكم تتعامل مع حدث النقر (Click). فإذا قام المستخدم بالنقر على نموذج، سوف يجرى تنفيذ الكود الموجود فى إجراء معالجة حدث Click بالنموذج. وأهم الأدوات التى تتسبب فى وقوع الأحداث هى لوحة المفاتيح والماوس. كما قد يتسبب التطبيق أو النظام فى وقوع أحداث داخلية لا يتدخل فى حدوثها المستخدم.

أحداث الماوس ولوحة المفاتيح

هناك عدد من الأحداث المتعلقة باستخدامات الماوس ولوحة المفاتيح. وكل حدث من هذه الأحداث يرتبط بإجراء معالجة يمكن كتابة كود به فى تطبيقات الويندوز. أهم هذه الأحداث هى:

- MouseDown. يحدث عند الضغط على زر الماوس أثناء وجود مؤشر الماوس فوق أداة التحكم.
- MouseLeave. يحدث عندما يترك مؤشر الماوس أداة تحكم.
- MouseEnter. يحدث عند دخول مؤشر الماوس إلى المتحكم.
- MouseMove. يحدث عند تحريك مؤشر الماوس فوق متحكم (Control).
- MouseUp. يحدث عند ترك الضغط على زر الماوس إثناء وجود مؤشر الماوس فوق متحكم.
- MouseHover. يحدث عندما يحوم مؤشر الماوس فوق المتحكم.
- KeyPress. يحدث عند الضغط على مفتاح أثناء اختيار أداة التحكم.
- KeyDown. يحدث عندما يكون المفتاح إلى أسفل أثناء اختيار أداة التحكم.
- KeyUp. يحدث عندما يكون المفتاح إلى أعلى أثناء اختيار أداة التحكم.

وتستقبل إجراءات معالجة أحداث الماوس معامل من تصنيف MouseEventArgs

يشتمل على بيانات خاصة بأحداث الماوس. بينما تستقبل إجراءات معالجة أحداث لوحة المفاتيح معامل من تصنيف KeyEventArgs يحتوى على بيانات تتعلق بأحداث لوحة المفاتيح.

التعرف على مفتاح المساعدة المستخدم

عند تكوين أحد التطبيقات التى تتعامل مع ضربات لوحة المفاتيح، قد نحتاج إلى مراقبة استخدام مفاتيح التعديل (Modifier keys)، مثل مفاتيح SHIFT، ALT، و CTRL. عند الضغط على مفتاح تعديل بالتزامن مع الضغط على مفتاح أخر، أو مع النقر على زر بالماوس، يمكن أن يترتب على ذلك قيام التطبيق بالإجراء المناسب. على سبيل المثال، عند الضغط على الحرف P سوف يظهر كما هو على شاشة الكمبيوتر، بينما يترتب على ضغط حرف P مع مفتاح CTRL طباعة المستند الجاري استخدامه.

ويمكن تحديد مفتاح التعديل الذى تم الضغط علية بتنفيذ الخطوات التالية:

- نستخدم عامل AND مع خاصية ModifierKeys ومع قيمة من تعداد Keys لمعرفة مفتاح التعديل الذى تم الضغط علية، كما يتضح من الكود التالى:

```
Private Sub button1_KeyPress (ByVal sender As Object, ByVal e As _
System.Windows.Forms.KeyPressEventArgs) Handles button1.KeyPress
  If (Control.ModifierKeys And Keys.Shift) = Keys.Shift Then
    MessageBox.Show ("Pressed " & Keys.Shift)
  End If
End Sub
```

- عند تنفيذ الكود السابق ثم النقر على زر button1_KeyPress أثناء الضغط على مفتاح SHIFT، سوف نحصل على رسالة بها رقم المفتاح المستخدم.

إجراءات معالجة احداث نماذج الويندوز

إجراء معالجة الحدث هو إجراء يتكون من الكود الذى نقوم بكتابته ويحدد التصرفات التى يتم تنفيذها عند وقوع أحد الأحداث، مثل قيام المستخدم بالنقر على زر أو استقبال طابور من الرسائل. وعندما يقع الحدث، يتم تنفيذ إجراء المعالجة الذى يتلقى الحدث. ويمكن تخصيص الحدث لعدد من إجراءات المعالجة، كما يمكن تغيير الوسائل التى تعالج الحدث ديناميكيا. الكود التالى يعرض صيغة إجراء معالجة حدث النقر على متحكم button:

```
Private Sub button1_Click (ByVal sender As System.Object, ByVal e As System.EventArgs) Handles button1.Click
End Sub
```

يحتوى هذا الإجراء على معاملين. المعامل الأول هو (Sender) الذى يحتوى على مرجع إلى الكائن مصدر الحدث. والمعامل الثانى هو (e) الذى يعتبر كائن من تصنيف EventArgs، ويحتوى على معلومات عن الحدث الذى وقع. ويمكن باستخدام خصائص هذا الكائن، الحصول على معلومات مثل موقع مؤشر الماوس بالنسبة للإحداث الخاصة بالماوس أو البيانات التى يتم تحويلها فى أحداث drag-and-drop. ويختلف الكائن الذى يمثله المعامل الثانى فى إجراء معالجة الحدث بين حدث وأخر. وبعض إجراءات المعالجة، مثل تلك المتعلقة بحدث MouseDown وحدث MouseUp، تحتوى على نفس الكائن فى المعامل الثانى بها. مثل هذا النوع من الأحداث، يمكن أن نستخدم معه نفس إجراء المعالجة. ويمكن أيضا، استخدام نفس إجراء المعالجة لمعالجة نفس الحدث فى أدوات تحكم مختلفة. على سبيل المثال، إذا كان لدينا مجموعة من أزرار الراديو على أحد النماذج، يمكن تكوين إجراء معالجة واحد لحدث النقر ثم ربط هذا الإجراء بحدث النقر فى جميع أدوات التحكم.

تكوين إجراءات معالجة الأحداث فى مصمم النماذج

تعتمد جميع تطبيقات الويندوز تقريبا على الاستجابة لأحداث المستخدم وأحداث النظام. ويمكننا تكوين إجراءات معالجة الأحداث فى مصمم نماذج الويندوز (Windows Form Designer) باستخدام القوائم المنسدلة التى تخص Class Name و Method Name بمربع تحرير الكود (Code Editor) فى Visual Studio.

لإيضاح كيفية إنشاء معالج حدث بهذه الطريقة، ننفذ الخطوات التالية:

١. ننقر بزر الماوس الأيمن على النموذج فى مصمم النماذج، ثم نختار View Code من القائمة المختصرة.

٢. من القائمة المنسدلة فى مربع Class Name فى أعلى يسار محرر الكود، نختار التصنيف الذى نريد إضافة إجراء معالجة حدث إلية.

٣. من القائمة المنسدلة فى مربع Method Name فى أعلى يمين محرر الكود، نختار الحدث الذى نريد تكوين معالج له. يترتب على ذلك، قيام Visual Studio

بإضافة إجراء معالجة للحدث ووضع مؤشر الكتابة داخل ذلك الإجراء، كما هو واضح من المثال السابق الخاص بالنقر على متحكم Button.

٤. يقوم المبرمج بعد ذلك بإدخال كود المعالجة الضروري داخل الإجراء الذى تم تكوينه.

تكوين إجراءات معالجة الأحداث الافتراضية

يؤدى النقر المزدوج على النموذج أو على أحد أدوات التحكم به إلى تكوين إجراء معالجة للحدث الإفتراضى. لتوضيح ذلك، ننفذ الخطوات التالية:

١. ننقر نقرا مزدوجا على النموذج أو أداة التحكم التى نريد تكوين إجراء معالجة للحدث الإفتراضى بها. يترتب على ذلك، فتح محرر الكود ووضع نقطة الإدراج داخل معالج الحدث الذى يأخذ الصيغة التالية:

```
Private Sub Button1_Click (ByVal sender As Object, ByVal e As EventArgs)
Handles Button1.Click

End Sub
```

٢. نضيف الكود المناسب داخل معالج الحدث السابق.

تكوين إجراءات معالجة الأحداث فى وقت التشغيل

يمكننا Visual Basic .NET من تكوين إجراءات معالجة الأحداث فى وقت التشغيل أيضا باستخدام الكود. يسمح لنا ذلك بالربط بين الحدث وبين إجراء المعالجة بناءا على توفر شروط معينة فى وقت التشغيل، بدلا من تنفيذ هذا الربط تلقائيا عند بدء تشغيل التطبيق. و لكى نقوم بتنفيذ ذلك، نتبع الخطوات التالية:

١. نفتح النموذج الذى نريد إضافة إجراء معالجة إلية، فى محرر الكود(Code Editor).

٢. نضيف إجراء إلى النموذج يحتوى على متطلبات الحدث الذى نريد معالجته. على سبيل المثال، عندما نريد إضافة إجراء معالجة حدث النقر على أحد الأزرار، نضيف كود مماثل لما يلى:

```
Private Sub Button1_Click (ByVal sender As Object, ByVal e As EventArgs)
End Sub
```

٣. نضيف كود المعالجة المناسبة فى داخل الإجراء.

٤. نحدد النموذج أو المتحكم الذى نريد تكوين معالج حدث خاص به.

٥. فى أحد الوسائل داخل تصنيف النموذج، نضيف الكود الذى يقوم باستدعاء وسيلة AddHandler المناسبة للحدث الذى نريد معالجته. على سبيل المثال، إذا كان الهدف هو معالجة حدث Click الخاص بمتحكم Button، سوف نضيف كود مماثل للكود التالى:

```
AddHandler Button1.Click, New System.EventHandler (AddressOf Me.Button1_Click)
```

ربط عدد من الأحداث بإجراء معالجة واحد

عند تصميم أحد التطبيقات، قد نجد أنة من الضروري إنشاء إجراء معالجة حدث واحد يتعامل مع عدد من الأحداث أو أن عدد من الإجراءات تكون مصدرا لنفس الحدث. على سبيل المثال، يمكن أن يكون أحد الأوامر فى قائمة مصدرا لنفس الحدث الذى يقع من أحد الأزرار على النموذج عندما يقوم الكائنان بنفس الوظيفة. وللقيام بعملية ربط عدد من الأحداث بإجراء واحد، نستخدم الكلمة المرشدة Handles مع القائمة المنسدلة الخاصة بأسماء التصنيفات والقائمة المنسدلة الخاصة بأسماء الوسائل فى محرر الكود، كما يتضح من الخطوات التالية:

١. ننقر بزر الماوس الأيمن على النموذج ونختار View Code.

٢. من القائمة المنسدلة فى مربع Class Name، نختار أحد التصنيفات الذى نريد ربطة بإجراء المعالجة.

٣. من القائمة المنسدلة فى مربع Method Name، نختار الحدث الذى نريد معالجته. يترتب على ذلك قيام محرر الكود بإدراج إجراء معالجة للحدث ووضع مؤشر الكتابة به، كما يتضح من الكود التالى الذى يعرض صيغة معالج حدث النقر على الأزرار:

```
Private Sub Button1_Click (ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
End Sub
```

٤. نلحق الأحداث بهذا الإجراء عن طريق فقرة handles، كما يتضح من الكود التالى:

```
Private Sub Button1_Click (ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click, Button2.Click
End Sub
```

٥. نضيف الكود المناسب إلى إجراء المعالجة.

مربعات الحوار فى نماذج الويندوز

تستخدم مربعات الحوار فى التفاعل مع المستخدم واستخراج المعلومات. ويمكن تعريف مربع الحوار بأنة نموذج تم ضبط خاصية FormBorderStyle به على القيمة FixedDialog. ويمكننا استخدام مصمم النماذج لتفصيل مربعات حوار تتوافق مع احتياجاتنا عن طريق إضافة أدوات تحكم (Controls)، مثل متحكم التمييز (Label)، مربع النص (TextBox)، أزرار الأوامر (Buttons). وبالإضافة إلى مربعات الحوار التى نقوم بتكوينها، يحتوى نظام NET Framework على مربعات حوار جاهزة يمكن استخدامها فى تطبيقاتنا، مثل مربع فتح الملفات، مربعات الرسائل.

تكوين مربعات الحوار

يمكننا تكوين مربع حوار فى وقت التصميم باستخدام الخطوات التالية:

- نضيف نموذج إلى المشروع بالنقر بزر الماوس الأيمن على المشروع فى مربع Solution Explorer، نشير إلى Add ثم ننقر على Window Forms.
- ننقر بزر الماوس الأيمن على النموذج الذى تم إضافته فى Solution Explorer ثم نختار Rename. نعيد تسمية النموذج ليكون "DialogBox..vb".
- فى نافذة Properties، نغير خاصية FormBorderStyle إلى FixedDialog.
- نقوم بتعديل شكل النموذج بما يتطابق مع احتياجاتنا.
- نجعل خاصية MaximizeBox، MinimizeBox، ControlBox تساوى False. لأن مربعات الحوار لا تشمل فى العادة شريط للقائمة (Menu Bar)، شرائط تدرج (Scroll Bar)، أزرار Minimize و Maximize، شريط مهام (Task Bar)، أو حدود قابلة للتغيير.
- نستخدم محرر الكود لإعداد إجراءات المعالجة المختلفة التى نحتاج إليها.

عرض مربعات الحوار

نعرض مربع الحوار بنفس الطريقة التى نعرض بها أى نموذج أخر فى التطبيق. بعد عرض نموذج بدء التشغيل تلقائيا، يمكننا تحميل وعرض أى نموذج أخر أو مربع حوار

بكتابة الكود اللازم لذلك. وبالمثل نستخدم الكود لإخفاء النموذج أو مربع الحوار وإزالته من الذاكرة. لتوضيح طريقة عرض مربع حوار بالتطبيق، ننفذ الخطوات التالية :

١. نفتح مربع تحرير الكود (Code Editor) ثم ننتقل إلى الإجراء الذى نريد استخدامه فى فتح مربع الحوار. يمكن ربط ذلك بأمر فى قائمة، النقر على أحد الأزرار، أو وقوع حدث أخر.

٢. فى إجراء معالجة الحدث، نضيف الكود اللازم لفتح مربع الحوار. المثال التال، يستخدم حدث النقر على متحكم Button لإظهار مربع الحوار :

```
    Private Sub Button1_Click (ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
        Dim dlg1 as new Form ()
        dlg1.ShowDialog ()
    End Sub
```

إدخالات المستخدم إلى مربعات الحوار

الوظيفة الرئيسية لمربعات الحوار هى حث المستخدم على إدخال البيانات التى يحتاجها التطبيق. ويقوم النموذج الذى يعرض مربع الحوار بمعالجة هذه المعلومات. ومن المهم معرفة كيفية إقفال مربع الحوار بعد عرضة. لمعرفة ذلك، يمكن استخدام قيمة خاصية DialogResult فى النموذج الأصلى الذى يعرض مربع الحوار، لتحديد أن المستخدم قد قام بالنقر على زر OK أو زر Cancel. وبناءا على قيمة خاصية DialogResult العائدة، يمكن معرفة ما إذا كانت هناك حاجة لاسترجاع معلومات من مربع الحوار. وفى حالة عدم استخدام الأزرار فى مربع الحوار وظهور الحاجة إلى إعادة قيمة لخاصية DialogResult، يمكن ضبط هذه الخاصية باستخدام الكود داخل مربع الحوار.

وعند عرض نموذج فى صورة مربع حوار Modal، يؤدى النقر على زر Close الذى يمثله حرف X فى أعلى يمين النموذج، إلى إخفاء النموذج وضبط خاصية DialogResult على القيمة DialogResult.Cancel. ولا يتم استدعاء وسيلة Close تلقائيا عندما يقوم المستخدم بالنقر على زر Close فى مربع الحوار أو ضبط قيمة خاصية DialogResult. وبدلا من ذلك، يتم إخفاء مربع الحوار ويمكن إعادة عرضة مرة أخرى بدون تكوين مثل جديد منة. ولهذا السبب، يجب استدعاء وسيلة Dispose بنموذج مربع الحوار عندما لا يعد التطبيق فى حاجة إلية.

الاحتفاظ بالإدخالات بعد إقفال مربعات الحوار

الطريقة التى يتم بها إغلاق مربع الحوار أو النتائج التى يمكن الحصول عليها منة، يمكن ضبطها فى وقت التصميم أو وقت التشغيل. فى وقت التصميم يمكن ضبط خاصية DialogResult لكل الأزرار الموجودة على سطح مربع الحوار. ويسمح لنا ضبط خاصية DialogResult وقت التشغيل بمعالجة استجابة المستخدم ديناميكيا. لضبط هذه الخاصية وقت التصميم لمتحكم Button، نتبع الخطوات التالية:

١. ننقر على متحكم Button على سطح مربع الحوار.

٢. نختار خاصية DialogResult فى نافذة Properties.

٣. نختار النتيجة المطلوبة من القائمة المنسدلة على يمين خاصية DialogResult.

ولضبط خاصية DialogResult لأداة تحكم أو نموذج باستخدام الكود:

١. ننتقل إلى إجراء معالجة الحدث أو الوسيلة التى نريد استخدامها لضبط خاصية DialogResult.

٢. ندخل الكود التالى بالخاصية:

```
Public Sub InformationProcessed ()
          لضبط خاصية DialogResult فى نموذج:
   Me.DialogResult = DialogResult.Yes
          لضبط خاصية DialogResult لأحد الأزرار:
   Button1.DialogResult = DialogResult.No
End Sub
```

ومع أن ضبط خاصية DialogResult بنموذج مربع الحوار فى المثال السابق، سوف تؤدى إلى إغلاق مربع الحوار تلقائيا، إلا أننا نستطيع الاستمرار فى معالجة حدث النقر قبل الإقفال. ويمكن استخدام الكود بإجراء المعالجة المذكور لإيقاف تنفيذ إغلاق مربع الحوار. لمنع خاصية DialogResult من إقفال مربع الحوار، ندخل الكود التالى فى إجراء معالجة حدث Click السابق:

```
Private Sub Button1_Click (ByVal sender As System.Object, ByVal e As
System.EventArgs)
    Me.DialogResult = DialogResult.None
End Sub
```

استخراج نتائج مربعات الحوار

بمجرد إقفال مربع الحوار، يمكن للنموذج الذى قام بعرضه الحصول على نتيجة التنفيذ عن طريق استخدام خاصية DialogResult أو من القيمة العائدة نتيجة استدعاء وسيلة ShowDialog بالنموذج. وبناءا على النتيجة أو القيمة العائدة، تكون استجابة النموذج الذى قام بعرض مربع الحوار. للحصول على قيمة خاصية DialogResult، نضيف كود مماثل للكود التالى فى الوسيلة التى تعرض مربع الحوار. ومن المفضل وضع هذا الكود بعد الكود الذى يقوم بتكوين وعرض مربع الحوار:

```
Public Sub DisplayDialog ()
  Dim dlg as New Form ()
  If dlg.ShowDialog = DialogResult.OK Then

  End If
End Sub
```

استخدام خصائص مربعات الحوار للحصول على المعلومات

من الطرق المعتادة للحصول على معلومات من مربع الحوار، استخدام مجموعة من الخصائص التى تعيد بنود منفردة من البيانات. يترتب على هذه الطريقة، استخراج البيانات التى نريد الحصول عليها فقط من مربع الحوار. للحصول على المعلومات بهذه الطريقة، نتبع الخطوات التالية:

١. فى التصنيف الذى نضع به كود مربع الحوار، نقوم بتكوين خصائص بالقدر الذى نحتاج إلية للحصول على المعلومات الخاصة بمربع الحوار.

٢. نضيف كود إلى إجراء Get فى تعريف الخاصية. ويمكن إزالة إجراء Set لمنع المستخدمين من تغيير معلومات مربع الحوار. يوضح الكود التالى كيفية الحصول على قيمة فى مربع قائمة مركب (Combo box)، يسمى cmbState من خلال خاصية بمربع الحوار:

```
Public Property StateSelected () As String
  Get
   Return cmbState.Text
  End Get
  Set (ByVal Value As String)
  End Set
End Property
```

وبمجرد تكوين جميع الخصائص التى نريد تواجدها لتوفير البيانات المطلوبة، يمكن الحصول على هذه البيانات بالتطبيق الذى يستخدم مربع الحوار. لتنفيذ ذلك، نتبع الخطوات التالية:

- فى النموذج الذى يقوم بعرض مربع الحوار، ننتقل إلى إجراء معالجة الحدث أو الوسيلة المستخدمة فى عرض مربع الحوار ونضيف كود يتعلق بجميع خصائص نموذج مربع الحوار، كما يتضح من المثال التالى:

```
Public Sub ShowMyDialog ()
  Dim Dlg as New Form2 ()
  Dlg.ShowDialog ()
  If Dlg.DialogResult = DialogResult.OK Then
    MessageBox.show Dlg.StateSelected
  End If
End Sub
```

الحصول على معلومات النموذج الأصلى لمربع حوار

فى التطبيقات التى نقوم بتكوينها، من الممكن أن نحتاج إلى المعلومات التى يوفرها النموذج الأصلى الذى بتبعة مربع الحوار. للحصول على معلومات من النموذج الأصلى فى مربع الحوار التابع له، نتبع الخطوات التالية:

١. نفتح تصنيف مربع الحوار.

٢. نستخدم خاصية Form.ParentForm فى كود مربع الحوار للوصول إلى الأعضاء العامة فى تصنيف النموذج الأصلى، كما يتضح من الكود التالى الذى يستخدم هذه الخاصية للوصول إلى وسيلة GetMyParentFormData العامة فى تصنيف النموذج الأصلى:

```
Public Sub GetParentData ()
  Dim x as String
  x = Me.ParentForm.GetMyParentFormData ()
End Sub
```

استخدام مربعات الرسائل

مربع الرسالة هو مربع من مربعات الحوار الموجودة بالنظام. ويقوم هذا المربع بعرض معلومات متعلقة بالتطبيق أمام المستخدم. كما تستخدم مربعات الرسائل أيضا فى طلب

معلومات من المستخدم (User).

لعرض معلومات أمام المستخدم فى مربع رسالة:

١. ننتقل إلى الإجراء الذى نريد إضافة كود مربع الرسالة به.

٢. نضيف كود يستخدم وسيلة MessageBox.Show، كما يتضح من الكود التالى الذى يوضح كيفية استدعاء وسيلة Show في تصنيف MessageBox لعرض معلومات أمام المستخدمين. وتستخدم وسيلة Show معامل Style الاختياري لتحديد نوع الأيقونة التى يتم عرضها فى مربع الرسالة:

```
Public Sub PerformCalculations ()
    MessageBox.Show ("The calculations are complete", "My _
Application", MessageBoxButtons.OKCancel, _
        MessageBoxIcon.Asterisk)
End Sub
```

ويمكن لمربعات الرسائل أن تستقبل إدخالات المستخدمين. كما أن وسيلة Show بتصنيف MessageBox تعيد قيمة يمكن استخدامها لمعرفة اختيار المستخدم. ويمكن تخزين هذه القيمة فى صيغة رقم صحيح (Integer) أو مقارنة القيمة العائدة باستخدام عبارة If. كما يمكن استخدام معامل Style بوسيلة Show لعرض الأزرار المناسبة للمعلومات التى يتم عرضها.

لعرض مربع رسالة لطلب معلومات:

١. نفتح محرر الكود الخاص بالتصنيف وننتقل إلى حيث نريد إضافة كود مربع الرسالة.

٢. نضيف كود يستخدم وسيلة Show لعرض مربع رسالة، كما يتضح من الكود التالى الذى يوضح كيفية استخدام مربع الرسالة فى استخراج معلومات من المستخدم وتحديد القيمة التى تم اختيارها:

```
Public Sub ExitApplication ()
    If MessageBox.Show ("Do you want to exit?", "My Application", _
      MessageBoxButtons.YesNo, MessageBoxIcon.Question) _
      = DialogResult.Yes Then
    Application.Exit
  End If
End Sub
```

ويمكن استخدام دالة MsgBox() التي كانت مستخدمة في الإصدارات السابقة من Visual Basic ، لتكوين مربع رسالة وعرضها أمام المستخدم. الكود التالى يوضح استخدام هذه الدالة :

```
Public Sub ExitApplication ()
  If MsgBox ("Do you want to exit?", MsgBoxStyle.Exclamation, _
"My Application") = MsgBoxResult.Yes Then
    Application.Exit ()
  End If
End Sub
```

أسلوب بناء البيانات فى نماذج الويندوز

يتم عرض البيانات فى نماذج الويندوز عن طريق الربط بين أدوات التحكم على سطح النموذج وبين مصادر البيانات. هذا الربط يمكننا من عرض وتغيير المعلومات الواردة من مصادر البيانات. ويمكننا Visual Basic .NET من استخدام مصادر البيانات التقليدية وأي هيكل بيانات أخر بنماذج الويندوز.

ربط البيانات فى نماذج الويندوز

فى نماذج الويندوز، يمكن الربط بين مع أى هيكل يحتوى على بيانات بغض النظر عن كيفية وصول البيانات إلى ذلك الهيكل. على هذا الأساس، يمكن الربط مع مصفوفة من القيم التى تنتج فى وقت التشغيل، أحد الملفات، وقيم أدوات تحكم أخرى. ويمكن ربط خصائص أدوات التحكم مع مصادر البيانات، مثل ضبط الرسم الخاص بأداة تحكم، ضبط لون خلفية أدوات التحكم، وضبط أحجام أدوات التحكم. بمعنى أخر، أصبح ربط البيانات طريقة أتوماتيكية لضبط أى خاصية بأحد أدوات التحكم على نموذج فى وقت التشغيل.

أنواع ربط البيانات فى نماذج الويندوز

هناك نوعان من أنواع ربط البيانات فى نماذج الويندوز:

- الربط البسيط للبيانات، الذى يعنى ربط أداة التحكم مع عنصر واحد من عناصر البيانات، مثل قيمة فى عمود بأحد جداول البيانات. ويلائم هذا النوع من الربط أدوات تحكم، مثل متحكم TextBox و متحكم Label. وهى أدوات تعرض عنصرا واحدا من عناصر البيانات. وعموما يمكن ربط أى خاصية فى أداة تحكم مع حقل فى قاعدة بيانات.

- الربط المركب، الذى يعنى قدرة المتحكم على الربط مع أكثر من عنصر من عناصر البيانات، مثل الربط مع أكثر من سجل فى قاعدة بيانات. ومن أمثلة أدوات التحكم التى تقبل هذا النوع من الربط، متحكم ListBox، متحكم DataGrid.

المهام الشائعة التى تستخدم ربط البيانات

عند التمعن فى تطبيقات الويندوز، سوف نجد أن معظم التطبيقات التجارية تستخدم قراءة المعلومات من نوع أو أخر من مصادر البيانات، ويتم ذلك فى الغالب عن طريق ربط البيانات. فيما يلى نذكر بعض المهام الشائعة التى تستخدم ربط البيانات للقيام بمهام عرض وتعديل هذه البيانات:

- إعداد التقارير (Reporting). حيث توفر التقارير طريقة مرنة لعرض وتلخيص البيانات فى مستند مطبوع. ومن الشائع جدا إنشاء تقرير يقوم بطباعة محتويات يجرى انتقاؤها من مصدر بيانات إما على الشاشة أو على الطابعة. وتشمل التقارير الشائعة: القوائم، الفواتير، والملخصات. ويتم صياغة البنود فى صورة أعمدة بالقوائم، وتنظيم البنود الفرعية تحت بند رئيسي.
- إدخال البيانات (Data Entry). من الطرق الشائعة لعرض البيانات وحث المستخدمين على إدخال المعلومات المطلوبة، استخدام نماذج إدخال البيانات. استخدام هذه النماذج يمكن المستخدم من إدخال المعلومات، أو اختيار البدائل باستخدام مربعات النصوص، أزرار الخيارات، القوائم المنسدلة، ومربعات الاختيار. وتعتبر هذه النماذج مصدرا للبيانات التى يتم تخزينها فى قاعدة بيانات.
- عرض واستخدام علاقات الأصل والتابع (Master/Detail Relationship) بين البيانات. على سبيل المثال، يمكن أن تكون هناك علاقات بين جدول رئيسي مثل جدول العملاء وبين جدول تابع هو جدول أوامر العملاء يحتوى على الأوامر التى تخص كل عميل.
- استخدام جداول الاستطلاع (Lookup Tables). حيث يمثل جدول الاستطلاع إحدى طرق عرض ومعالجة البيانات عن طريق الربط الضمنى بين الأسماء المألوفة التى يجرى عرضها غالبا فى مربعات سرد مركبة وبين بيانات مخزنة فى قواعد البيانات.

تكوين أداة ربط بيانات على نموذج الويندوز

يسمح لنا الربط البسيط بعرض عنصر بيانات منفرد، مثل قيمة عمود فى جدول بفئة بيانات، فى أداة تحكم على سطح نموذج. ويمكن فى Visual Basic، الربط البسيط بين أى خاصية أداة تحكم وبين عنصر بيانات.

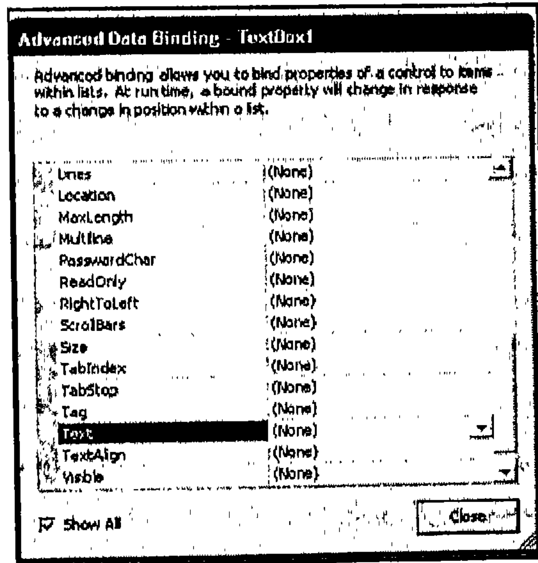
للقيام بالربط البسيط مع أداة تحكم، ننفذ الخطوات التالية:

١. نجرى اتصال مع مصدر البيانات.

٢. نختار أداة تحكم على النموذج لعرض مربع Properties الخاص به.

٣. نقوم بتوسيع خاصية DatatBindings، مما يؤدى إلى عرض الخصائص الشائع الربط معها. على سبيل المثال، سوف تظهر خاصية Text مع معظم أدوات التحكم.

٤. وعندما لا تكون الخاصية التى نريد ربطها من بين الخصائص الشائع الربط معها، ننقر علامة الحذف (Ellipsis) فى مربع Advanced لعرض مربع حوار Advanced Data Binding مع قائمة كاملة بخصائص المتحكم، كما يتضح من الشكل رقم (١١) الذى يعرض خصائص TextBox.



شكل رقم ١١

٥. ننقر سهم القائمة المنسدلة للخاصية التى نريد الربط معها، مما يترتب علية عرض قائمة بمصادر البيانات المتاحة للربط.

٦. نقوم بتوسيع مصدر البيانات الذى نريد الربط معه ثم ننتقل إلى العنصر الذى نريد ربطة. على سبيل المثال، عندما نريد الربط مع قيمة عمود فى جدول بفئة بيانات (Dataset)، نقوم بتوسيع اسم فئة البيانات ثم توسيع اسم جدول بيانات لعرض أسماء الأعمدة.

٧. ننقر على اسم العنصر الذى نريد الربط معه.

٨. بعد انتهاء العمل مع مربع حوار Advanced Data Binding، ننقر زر Close للعودة إلى نافذة Properties.

٩. عندما نريد ربط خصائص أخرى بأداة التحكم، نكرر الخطوات من ٣ إلى ٧.

مصادر توريد البيانات لنماذج الويندوز

من المعتاد استخدام ربط البيانات فى التطبيقات للاستفادة من البيانات المخزنة فى قواعد البيانات. بالإضافة إلى ذلك، تسمح لنا نماذج الويندوز بالربط مع البيانات فى هياكل بيانات أخرى، مثل المصفوفات (Arrays) والمجموعات (Collections). تلخص لنا النقاط التالية أنواع هياكل البيانات التى يمكن الربط معها، كما تقدم ملاحظات عن خصائص هذا الربط.

- المصفوفة أو المجموعة (Array or Collection)، التى تنفذ واجهة بينية من نوع IList . من الأمثلة على ذلك، مصفوفات تصنيف System.Array.
- كائنات بيانات ADO.NET. حيث توفر ADO.NET عددا من هياكل البيانات المناسبة للربط معها. وتختلف هذه الهياكل فى درجة تعقيدها.
- كائن DataColumn يعتبر اللبنة الأساسية فى بناء كائن DataTable لأن الجدول يتكون من عدد من الأعمدة. وكل كائن من كائنات DataColumn يحتوى على خاصية DataType التى تحدد نوع البيانات التى يحتفظ بها العمود. ويمكن الربط البسيط بين خاصية فى أداة تحكم ، مثل خاصية Text، وبين عمود فى جدول بيانات.

- كائن DataTable. يمثل هذا الكائن جدول بيانات يحتوى على صفوف وأعمدة. ويحتوى جدول البيانات على مجموعتين: مجموعة DataColumn، التى تمثل البيانات فى الجدول، ومجموعة DataRow، التى تمثل صفوف البيانات فى الجدول. وبمكن الربط المركب بين جدول البيانات وبين أداة تحكم، مثل شبكة البيانات (Data Grid).
- كائن DataView. يمثل هذا الكائن مشهدا يعدة المستخدم طبقا لاحتياجاته من جدول بيانات مفرد بعد ترشيحه أو فرزة. ويمكن الربط المركب مع مشهد البيانات، كما يمكن الربط البسيط مع بيانات داخل المشهد. ويجب ملاحظة أن مشهد البيانات يمثل صورة ثابتة من جدول البيانات.
- كائن Dataset. وهو مجموعة من الجداول، العلاقات، وقيود البيانات فى قاعدة بيانات. ويمكن استخدام هذا الكائن فى الربط المركب أو الربط البسيط.
- كائن DataViewManager. هو مشهد لكامل فئة البيانات يناظر كائن DataView بالإضافة إلى احتوائه على علاقات. وتسمح لنا مجموعة DataViewSettings بضبط المرشحات وخيارات الفرز لأى مشهد بيانات (DataView) متعلق بأحد الجداول فى مدير مشاهد البيانات (DataViewManager).

إدارة ربط البيانات بنماذج الويندوز

يرتبط باستخدام مصادر البيانات للربط مع أدوات التحكم بنماذج الويندوز، استخدام كائنات من تصنيف CurrencyManager للقيام بتتبع الربط مع مصادر البيانات والأشراف علية. ويوجد بالنموذج كائن CurrencyManager لكل مصدر بيانات منفصل يتم الربط معه. وإذا كانت جميع أدوات التحكم على النموذج مرتبطة بمصدر بيانات واحد، فإن جميع هذه الأدوات سوف تشترك فى كائن CurrencyManager واحد. من ناحية أخرى، هناك أوقات تكون فيها أدوات التحكم الموجودة على النموذج مرتبطة مع مصادر بيانات مختلفة. فى هذه الحالة، يكون هناك كائنات CurrencyManager متعددة على النموذج، كل واحد منها يتتبع الارتباط مع أحد مصادر البيانات. ويتم إدارة جميع كائنات CurrencyManager الموجودة على النموذج باستخدام كائن من تصنيف BindingContext. يتضح من هذا البناء أن كل نموذج ويندوز يحتوى على أدوات تحكم مرتبطة بمصادر بيانات سوف يكون به

على الأقل كائن من تصنيف BindingContext، يقوم بإدارة واحد أو أكثر من كائنات CurrencyManager التى تدير بدورها الربط مع مصادر البيانات.

تصنيف CurrencyManager

يستخدم هذا التصنيف للاحتفاظ بأدوات التحكم المرتبطة بالبيانات متزامنة مع بعضها البعض. ويقوم تصنيف CurrencyManager بهذه المهمة عن طريق إدارة مجموعة من البيانات المرتبطة الواردة من مصدر بيانات واحد. ويحتفظ النموذج بكائن CurrencyManager لكل مصدر بيانات مرتبط مع النموذج. وبالنظر إلى احتمال وجود أكثر من مصدر بيانات مرتبط مع النموذج، يقوم كائن أخر هو BindingContext بإدارة كائنات CurrencyManager الموجودة على النموذج. وبعبارة أخرى، كل كائنات التحكم فى حاوية من الحاويات، يكون لها على الأقل كائن BindingContext واحد يقوم بإدارة كائنات CurrencyManager الموجودة بها. وعلى خلاف التقنيات السابقة للوصول البيانات، مثل تقنية ADO، لا يحتفظ ADO.NET بمؤشر يدل على موقع السجل الحالى داخل البيانات. ولهذا نجد أن خاصية Position فى تصنيف CurrencyManager، لها أهمية خاصة لقيامها بوظيفة تحديد الوضع الحالى لكل أدوات التحكم المرتبطة بنفس كائن CurrencyManager.

تصنيف BindingContext

يحتوى كل نموذج على الأقل على واحد من كائنات BindingContext التى تدير الكائنات من نوع CurrencyManager الموجودة على النموذج. ويمكننا هذا الكائن من استخراج أى كائن من كائنات CurrencyManager مرتبط بمصدر بيانات. على سبيل المثال، عندما نقوم بربط مربع نص على النموذج مع عمود فى جدول بيانات بأحد فئات البيانات (Dataset)، فإن مربع النص يتصل مع كائن BindingContext على النموذج، الذى يقوم بدورة بالاتصال مع كائن CurrencyManager المتعلق بالبيانات المطلوبة. وعند الاستعلام عن خاصية Position فى كائن CurrencyManager، سوف نحصل على السجل الحالى فى البيانات المرتبطة مع مربع النص. فى المثال التالى، يتم ربط مربع نص مع عمود FirstName بجدول Customers فى فئة بيانات DataSet1 باستخدام كائن BindingContext الخاص بالنموذج:

```
Textbox1.DataBindings.Add ("Text", DataSet1, "Customers.FirstName")
```

ولتوسيع النموذج المذكور أعلاه، يمكننا إضافة مربع نص ثانى هو TextBox2، إلى

النموذج وربطة مع عمود LastName فى جدول Customers بنفس فئة البيانات. وبالنظر إلى أن كائن BindingContext على علم بالربط الأول، لذلك سوف يستخدم نفس كائن CurrencyManager لأن كلا من مربعي النص يرتبط مع نفس مصدر البيانات المتمثل فى فئة البيانات:

```
Textbox2.DataBindings.Add ("Text", DataSet1, "Customers.LastName")
```

وفى حالة ربط مربع نص TextBox2 مع فئة بيانات مختلفة، سوف يقوم BindingContext بتكوين وإدارة كائن CurrencyManager أخر. ومن المهم الثبات فيما يتعلق بضبط خاصية DataSource وخاصية DisplayMember حتى لا يقوم BindingContext بتكوين كائنات CurrencyManager متعددة لنفس فئة البيانات، مما ينتج عنة أخطاء. ويمكن استخدام إحدى الطرق التالية لضبط هذه الخصائص، فقط يجب التحقق من الثبات فى استخدام نفس الطريقة خلال الكود:

```
ComboBox1.DataSource = DataSet1
ComboBox1.DisplayMember = "Customers.FirstName"
Me.BindingContext (DataSet1, "Customers").Position = 1
```

أو الطريقة التالية:

```
ComboBox1.DataSource = DataSet1.Customers
ComboBox1.DisplayMember = "FirstName"
Me.BindingContext (DataSet1.Customers).Position = 1
```

التجول فى البيانات بنماذج الويندوز

يتم إدارة عملية التنقل بين السجلات فى مصدر بيانات فى تطبيق نماذج الويندوز، باستخدام طبقة ربط البيانات. ويرجع ذلك إلى أن كائن CurrencyManager المرتبط مع جدول فى فئة بيانات، يدعم خاصية Position التى تقوم بتحديد الموقع فى مصدر البيانات. ويستخدم نظام ربط البيانات هذه الخاصية للتحقق من وجود التزامن فى استخدام نفس السجل. ونظرا لأن فئة البيانات يمكن أن تحتوى على مصادر متعددة للبيانات، أو لأن أدوات التحكم على النموذج يمكن ربطها مع اثنين أو أكثر من مصادر البيانات، لذلك يمكن أن يكون هناك العديد من كائنات CurrencyManager على النموذج، واحد لكل مصدر بيانات. ولتبسيط عملية ربط البيانات، توفر نماذج الويندوز كائن BindingContext

لتوفير نقطة وصول واحدة إلى كائنات CurrencyManager. للتنقل داخل البيانات بنماذج الويندوز، نقوم بتنفيذ ما يلى:

- نضبط خاصية Position فى كائن CurrencyManager الخاص بمصدر البيانات المستخدم فى الربط، على موقع السجل الذى نريد الوصول إلية. فى المثال التالى، نقوم بزيادة الموقع فى كائن CurrencyManager بمقدار واحد عند النقر على زر nextButton. وفى هذا المثال، يرتبط كائن CurrencyManager مع جدول authors فى فئة البيانات dsPubs1.

```
Private Sub nextButton_Click (ByVal sender As Object, _
ByVal e As EventArgs) Handles nextButton.Click
  Me.BindingContext (dsPubs1, "authors").Position += 1
End Sub
```

ويجب ملاحظة أن ضبط خاصية Position على قيمة تتعدى السجل الأول أو السجل الأخير لن ينتج عنة خطأ، لأن نظام NET Framework لا يسمح بضبط قيمة موقع خارج حدود قائمة البيانات. ويمكن اختبار أن قيمة الموقع قد تعدت حدود القائمة، باستخدام الكود التالى الذى يحجب زر Next الموجود على النموذج عند الوصول إلى البند الأخير فى البيانات:

```
Protected Sub Binding_PositionChanged (ByVal sender As _
Object, ByVal e As EventArgs)
If Me.BindingContext (dsPubs1, "authors").Position =
Me.BindingContext(dsPubs1, "authors").Count - 1 Then
      nextButton.Enabled = False
   Else
      nextButton.Enabled = True
   End If
End Sub
```

تطبيقات واجهات الوثائق المتعددة

تسمح لنا تطبيقات واجهات الوثائق المتعددة (MDI) بعرض نوافذ متعددة فى نفس الوقت، كل منها يحتوى على إحدى الوثائق. وتتميز هذه التطبيقات بوجود قائمة للتنقل بين الوثائق المختلفة.

تكوين نماذج MDI الاصلية

يمثل النموذج الأصلى (Parent Form) الأساس فى تطبيق واجهات الوثائق المتعددة.

ويحتوى هذا النموذج على نوافذ الوثائق التابعة التى عن طريقها يتفاعل المستخدم مع التطبيق. ويعتبر تكوين النموذج الأصلى سهلا، سواء تم ذلك باستخدام مصمم النماذج أو باستخدام الكود. لتكوين نموذج أصل فى وقت التصميم:

١. نقوم بإنشاء تطبيق ويندوز جديد.

٢. نختار النموذج الإفتراضى. وفى نافذة Properties، نضبط خاصية IsMDIContainer على القيمة True. يترتب على ذلك، جعل النموذج حاوية MDI . ويمكن أيضا ضبط خاصية WindowState على Maximized عند الرغبة فى تكبير النموذج الأصلى إلى أقصى حد.

٣. من مربع ToolBox، نسحب مكون MainMenu إلى النموذج. نضيف بند قائمة فى المستوى الأعلى ونجعل خاصية Text تساوى &File ثم نضيف قوائم فرعية تسمى &Close، و &New. ونضيف أيضا بند قائمة فى المستوى الأعلى باسم &Window. سوف نستخدم القائمة الأولى فى إضافة وإخفاء بنود بالقائمة وقت التشغيل، والقائمة الثانية سوف تتابع النوافذ المفتوحة. بذلك يتم تكوين النافذة الأصلية.

٤. نضغط F5 لتشغيل التطبيق.

تكوين نماذج MDI التابعة

تعتبر النماذج التابعة فى تطبيقات واجهات الوثائق المتعددة، عنصرا ضروريا فى هذا النوع من التطبيقات. فى الإجراء التالى، سوف نقوم بتكوين نموذج MDI تابع لعرض متحكم RichTextBox، مماثل لمعظم تطبيقات معالجة الكلمات:

١. نكون نموذج MDI أصلى به قائمة تحتوى على بنود File، Window فى المستوى الأعلى من القائمة، وبنود New، و Close تابعة لقائمة File.

٢. فى القائمة المنسدلة فى قمة نافذة Properties، نختار بند قائمة Window ونضبط خاصية MDIList على القيمة True. يؤدى ذلك إلى احتفاظ قائمة Window بأسماء النوافذ التابعة ووضع علامة الفحص بجانب النافذة النشطة.

٣. فى مربع Solution Explorer، ننقر بزر الماوس الأيمن على المشروع، ونشير إلى Add، ثم ننقر Windows Form.

٤. فى مربع حوار Add New Item، نختار Local Project Items فى جانب Categories ونختار Windows Form فى الجانب الأيمن من مربع الحوار. ندخل اسما للنموذج فى مربع Name ثم ننقر زر Open لإضافة النموذج إلى المشروع. يترتب على ذلك فتح نافذة Windows Form Designer وعرض Form2 به.

٥. نسحب كائن RichTextBox من مربع ToolBox إلى النموذج.

٦. فى نافذة Properties، نضبط خاصية anchor على قيم Left و Top، وخاصية Dock على Fill. يترتب على ذلك شغل متحكم RichTextBox لكامل منطقة نموذج MDI التابع.

٧. ندخل الكود التالى فى إجراء معالجة النقر على بند قائمة New، لتكوين نماذج تابعة جديدة عند النقر علية:

```
Protected Sub MDIChildNew_OnClick (ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles MenuItem2.Click
    Dim NewMDIChild As New Form2 ()
    NewMDIChild.MDIParent = Me
    NewMDIChild.Show ()
End Sub
```

٨. نضغط F5 لتشغيل التطبيق.

تشغيل التطبيق والنقر على New فى قائمة File يؤدى إلى تكوين نماذج تابعة جديدة يمكن تتبعها فى قائمة Window. وعندما يحتوى نموذج MDI تابع على مكون MainMenu ثم يتم فتح هذا التابع داخل نموذج MDI أصلى به نفس المكون، سوف يقوم التطبيق بدمج بنود القوائم تلقائيا عندما نضبط خاصية MergType واختياريا خاصية MergOrder. ولكي يتم دمج بنود قائمتين، يجب ضبط خاصية MergType على القيمة MergeItems فى كل من القائمتين.

تحديد النموذج التابع النشط

فى بعض الأحيان قد نحتاج إلى استخدام أحد الأوامر الذى يتم تطبيقه على أداة تحكم تحوز التركيز فى نموذج تابع نشط. على سبيل المثال، نفترض أننا نريد نسخ نص من مربع نص على نموذج تابع إلى لوحة القص (Clipboard). ونفترض أننا سوف نستخدم إجراء

معالجة حدث النقر على أمر Copy فى قائمة Edit بالقائمة الرئيسية. لن يكون بالإمكان التنفيذ المباشر لهذا الأمر لأن تطبيق MDI يمكن أن يحتوى على العديد من النماذج التابعة. لذا يحتاج إجراء المعالجة المذكور إلى معرفة النموذج الواجب استخدامه. ولتحديد النموذج الصحيح، نستخدم خاصية ActiveMDIChild، التى تعيد النموذج التابع النشط. وعندما يكون هناك العديد من أدوات التحكم على النموذج، نحتاج أيضا إلى معرفة الأداة النشطة من بينها. ولتحقيق هذا الهدف، نستخدم خاصية ActiveControl التى تعيد إلينا الأداة التى تحوز التركيز على النموذج النشط. يوضح الكود التالى إجراء نسخ يمكن أن نستدعيه باستخدام بند فى قائمة أو زر على شريط أدوات:

```
Public Sub mniCopy_Click (ByVal sender As Object, _
  ByVal e As System.EventArgs) Handles mniCopy.Click
  Dim activeChild As Form = Me.ActiveMDIChild
  If (Not activeChild Is Nothing) Then
    Try
      Dim theBox As RichTextBox = _
      Ctype (activeChild.ActiveControl, RichTextBox)
      If (Not theBox Is Nothing) Then
          Clipboard.SetDataObject (theBox.SelectedText)
      End If
    Catch
      MessageBox.Show ("You need to select a RichTextBox.")
    End Try
  End If
End Sub
```

فى المثال السابق، يتم الحصول على النموذج التابع النشط. وعند التحقق من وجود هذا النموذج، يقوم البرنامج بالحصول على أداة RichTextBox النشطة بالنموذج النشط باستخدام عبارة:

```
Dim theBox As RichTextBox = Ctype (activeChild.ActiveControl, RichTextBox)
```

وعند التحقق من وجود هذه الأداة النشطة، يتم نسخ النص المختار بها إلى لوحة القص باستخدام العبارة التالية:

```
Clipboard.SetDataObject (theBox.SelectedText)
```

إرسال البيانات إلى النموذج التابع النشط

عند استخدام تطبيقات واجهة الاستخدام ذات الوثائق المتعددة، نحتاج غالبا إلى إرسال بيانات إلى النافذة التابعة النشطة، مثلما يفعل المستخدم عندما يقوم بلصق بيانات من لوحة القص فى تطبيق MDI. لتوضيح ذلك، نعرض المثال التالى الذى يفترض وجود نموذج MDI أصلى باسم Form1 يتبعة واحدا أو أكثر من النماذج التابعة التى تحتوى على أداة التحكم RichTextBox. نستخدم الكود التالى لنسخ النص الموجود على لوحة القص إلى المتحكم النشط على النموذج التابع النشط:

```
Public Sub mniPaste_Click (ByVal sender As Object, _
  ByVal e As System.EventArgs) Handles mniPaste.Click
  Dim activeChild As Form = Me.ActiveMDIChild
  If (Not activeChild Is Nothing) Then
    Try
      Dim theBox As RichTextBox = Ctype (activeChild.ActiveControl,
RichTextBox)
      If (Not theBox Is Nothing) Then
        Dim data As IDataObject = Clipboard.GetDataObject ()
        If (data.GetDataPresent (DataFormats.Text)) Then
          theBox.SelectedText = data.GetData (DataFormats.Text).ToString
()
        End If
      End If
    Catch
      MessageBox.Show ("You need to select a RichTextBox.")
    End Try
  End If
End Sub
```

تنسيق النماذج التابعة

توجد بتطبيقات واجهة الاستخدام ذات الوثائق المتعددة فى الغالب، قائمة تحتوى على أوامر تقوم بتنفيذ عمليات، مثل التعاقب (Cascade)، والترتيب (Arrange) للنماذج التابعة المفتوحة. لترتيب النماذج التابعة المفتوحة فى النموذج الأصلى، نستخدم وسيلة LayoutMDI مع تعداد MDILayout. حيث يمكن استخدام واحدة من القيم المختلفة فى

تعداد MDILayout بواسطة وسيلة LayoutMDI لترتيب النماذج التابعة المفتوحة بالتعاقب، أفقيا ورأسيا، أو فى شكل أيقونات مرتبة على طول الجزء الأسفل من النموذج الأصلى. ويتم فى الغالب، استخدام هذه الوسيلة داخل إجراء معالجة لحدث النقر على بند فى قائمة، مثل النقر على بند Cascade Windows. الكود التالى يستخدم قيمة Cascade بتعداد MDILayout لترتيب النماذج التابعة لنموذج Form1 الأصلى، وقد تم وضع الكود فى إجراء معالجة حدث النقر على بند Cascade Windows فى قائمة Window:

```
Protected Sub CascadeWindows_Click (ByVal sender As System.Object, ByVal e As
System.EventArgs)
  Form1.LayoutMDI (System.Windows.Forms.MDILayout.Cascade)
End Sub
```

استخدام القوائم فى نماذج الويندوز

تستخدم القوائم (Menus) والقوائم المختصرة (Context Menus) للكشف عن وظائف يقوم بها التطبيق أمام المستخدمين أو التنبيه عليهم بخصوص المعلومات المهمة داخل التطبيق. وتحتفظ القوائم بأوامر موضوعة فى مجموعات على أساس تعلقها بموضوع عام. وتطفو القوائم المختصرة عند النقر بزر الماوس الأيمن على النموذج أو أحد الكائنات، وتحتوى هذه القوائم على الأوامر المتعلقة بمنطقة معينة فى النموذج.

قوائم نماذج الويندوز

يمثل شريط القائمة (Menu) جزءا حاسما فى واجهة الاستخدام بتطبيقات الويندوز. ويبدأ العمل مع القوائم بإضافتها إلى النماذج وإضافة بنود إلى هذه القوائم.

إضافة القوائم وبنود القوائم إلى نماذج الويندوز

يتم تكوين القائمة بنموذج الويندوز عن طريق استخدام كائن MainMenu، الذى يحتوى على مجموعة من كائنات MenuItem. ويمكننا إضافة قوائم إلى نماذج الويندوز فى وقت تصميم التطبيق عن طريق إضافة مكون MainMenu إلى النموذج، ثم إلحاق بنود قائمة بة باستخدام مصمم القوائم (Menu Designer). ويمكن أيضا إضافة القوائم باستخدام الكود عن طريق إضافة واحد أو أكثر من كائنات MainMenu إلى نموذج الويندوز ثم إضافة كائنات MenuItem إلية. وسوف نوضح فيما يلى كيفية تكوين القوائم باستخدام مصمم القوائم وباستخدام الكود. لإضافة قائمة إلى نموذج ويندوز فى وقت التصميم :

١. نفتح النموذج الذى نريد إضافة قائمة إلية فى مصمم نماذج الويندوز.

٢. فى مربع ToolBox، نقوم بالنقر المزدوج على مكون MainMenu لإضافة مكون قائمة إلى النموذج.

لإضافة قائمة إلى نموذج ويندوز باستخدام الكود:

١. فى نافذة محرر الكود (Code Editor)، نكون وسيلة عامة جديدة لكى تشتمل على الكود اللازم لإضافة قائمة إلى النموذج، كما يتضح من الكود التالى:

```
Public Sub AddMenu ()
End Sub
```

٢. نضيف الكود التالى داخل الإجراء السابق لتكوين مثل من مكون MainMenu:

```
Dim mnuFileMenu as New MainMenu ()
Me.Menu = mnuFileMenu
```

بعد إضافة مكون MainMenu إلى نموذج الويندوز، نحتاج إلى إضافة بنود قائمة إلية. ويسمح لنا مصمم القوائم بإضافة هذه البنود وقت التصميم. ويتم الاحتفاظ بمكونات القائمة فى مجموعة (Collection)، ولذلك يمكن إضافة بنود إلى القائمة فى وقت التشغيل عن طريق إضافة كائنات MenuItem إلى المجموعة. لإضافة بنود إلى القائمة فى وقت التصميم:

١. ننقر على مكون MenuItem فى نموذج الويندوز. يترتب على ذلك، عرض نص "Type Here".

٢. ننقر على النص السابق، ونطبع اسم القائمة الذى نرغب فى إضافته.

٣. لإضافة بند أخر إلى القائمة، ننقر منطقة "Type Here" أخرى داخل مصمم القوائم، على الترتيب التالى:

أ. ننقر المنطقة التى علي يمين بند القائمة الحالى لإضافة قائمة فرعية.

ب. ننقر المنطقة التى أسفل بند القائمة الحالى لإضافة بند أخر فى نفس القائمة الفرعية.

لإضافة بنود إلى قائمة باستخدام الكود:

١. نضيف مكون MenuItem إلى النموذج باستخدام الكود التالى داخل إحدى الوسائل:

```
Public Sub AddMenuAndItems ()
  Dim mnuFileMenu as New MainMenu ()
  Me.Menu = mnuFileMenu
End Sub
```

٢. نكـون كائنات MenuItem لإضافتها إلى مجموعة كائن MainMenu باستخدام الكود التالى:

```
Dim myMenuItemFile as New MenuItem ("&File")
Dim myMenuItemNew as New MenuItem ("&New")
```

٣. نضيف بنود القائمة السابق تكوينها فى البند السابق إلى مجموعة البنود باستخدام الكود التالى:

```
mnuFileMenu.MenuItems.Add (myMenuItemFile)
myMenuItemFile.MenuItems.Add (myMenuItemNew)
```

لإضافة قوائم فرعية (SubMenus)، نضيف كائنات MenuItem إلى خاصية MenuItems فى كائن MenuItem الأصلى. فى المثال السابق، عندما نريد إضافة قائمة فرعية باسم myMenuItemFolder لبند myMenuItemNew، .نضيف الكود التالى:

```
myMenuItemNew.MenuItems.Add (myMenuItemFolder)
```

بعد الانتهاء من تكوين القائمة، سوف نحتاج إلى إضافة الوظائف التى تقوم بها بنود هذه القائمة. ويتم إنجاز ذلك من خلال إجراءات معالجة حدث MenuItem.Click.

تحريك بنود قوائم الويندوز

يمكن تحريك بنود القائمة بسهولة بسبب الاحتفاظ بها فى مجموعة. فى وقت التصميم، يمكننا تحريك هياكل قوائم بالكامل فى دورة داخل مصمم القوائم. وفى وقت التشغيل، يمكن تحريك بنود قائمة بين كائنات MainMenu أو بين كائنات MenuItem.

لتحريك قائمة فى وقت التصميم، نتبع الخطوات التالية:

١. فى مصمم القوائم، ننقر على بند القائمة المستهدف ثم نسحبة إلى الموقع الجديد. ويمكن وضعة فى قائمة القمة على طول شريط القائمة، إدراجه بين بنود قائمة أخرى، أو إضافته بصفة قائمة فرعية جديدة لبند قائمة موجود.

ولتحريك قائمة باستخدام الكود، نغير موقع البند على الفهرس، كما يتضح من الكود التالى:

```
Public Sub MoveMenuItemDown ()
Me.Menu.MenuItems (0).MenuItems (0).Index += 1
```

نسخ بنود قوائم الويندوز

عند تكوين تطبيقات نماذج الويندوز نحتاج أحيانا إلى الحصول على نسخة من قائمة سبق تكوينها. على سبيل المثال، يتم فى الغالب تكرار أوامر قائمة Edit التقليدية. ويمكننا نسخ القوائم وقت التصميم باستخدام مصمم القوائم، كما يمكن استخدام وسيلة CloneMenu للقيام بهذه المهمة باستخدام الكود.

لنسخ بنود قائمة فى وقت التصميم:

١. فى مصمم القوائم، نختار بند القائمة التى نريد تكرارها. ننقر بزر الماوس الأيمن عليها ثم نختار Copy من القائمة المختصرة.

٢. ننتقل فى داخل مصمم القوائم إلى المنطقة التى نريد تكرار القائمة بها. ننقر بزر الماوس الأيمن ثم نختار Paste لوضع القائمة فى الموقع الجديد.

ولنسخ بنود قائمة باستخدام الكود، نستخدم المثال التالى الذى يقوم بتكوين قائمة تحتوى على بند Edit فى القائمة الرئيسية. ويحتوى بند Edit على ثلاثة أوامر هى: Copy، Cut، و Paste. نقوم بعد ذلك بنسخ قائمة Edit إلى مكون ContextMenu باستخدام الكود التالى الذى يفترض أن النموذج المستخدم هو Form1.

```
Private mmMainMenu As MainMenu
Private miEditMenu As MenuItem
Private cmEdit As ContextMenu


Private Sub CopyMenuItems ()
  mmMainMenu = New MainMenu
  miEditMenu = New MenuItem ("&Edit")
  miEditMenu.MenuItems.Add ("&Cut")
  miEditMenu.MenuItems.Add ("&Copy")
  miEditMenu.MenuItems.Add ("&Paste")

  mmMainMenu.MenuItems.Add (miEditMenu)
  Form1.Menu = mmMainMenu

  cmEdit = New ContextMenu
  cmEdit.MenuItems.Add (miEditMenu.CloneMenu ())
  Form1.ContextMenu = cmEdit
```

```
End Sub
```

ويجب العلم بأن نسخ بنود قائمة يؤدى إلى الاحتفاظ بقيم الخصائص وإجراءات المعالجة التى سبق تكوينها مع بنود القائمة الأصلية.

حجب بنود القوائم

يعتبر حجب أو التمكين من بنود قائمة استجابة لأفعال المستخدم، طريقة لتحجيم أو توسيع الأوامر التى يمكن أن ينفذها المستخدم. فى الوضع الإفتراضى، تكون بنود القائمة متاحة للاستخدام عند تكوينها، ولكن يمكن تعديل ذلك من خلال خاصية Enabled. ويمكن التعامل مع هذه الخاصية فى وقت تصميم التطبيق باستخدام نافذة Properties، أو باستخدام الكود.

لحجب بند فى قائمة أثناء التصميم، نختار البند ثم نضبط خاصية Enabled على القيمة False فى مربع Properties.

ولحجب بند قائمة باستخدام الكود، نضبط خاصية Enabled داخل إجراء المعالجة المستخدم باستخدام الكود التالى:

```
MenuItem1.Enabled = False
```

ويجب ملاحظة أن حجب بند فى القائمة الرئيسية، يترتب علية حجب جميع البنود فى القائمة التابعة للبند الرئيس. كما أن حجب بند قائمة يتبعة قائمة فرعية، يترتب علية حجب جميع البنود التى تحتويها القائمة. وعند حجب جميع البنود فى قائمة، فإن من الممارسات الجيدة إخفاء القائمة بالكامل لتنظيف واجهة الاستخدام. ويجب العلم أن إخفاء القائمة وحدة لا يمنع استخدامها، بل يجب إخفاء وحجب القائمة لكى لا يستطيع المستخدم تنفيذ أوامرها باستخدام المفاتيح المختصرة.

إخفاء بنود القوائم

يعتبر إخفاء بنود قائمة وسيلة للتحكم فى واجهة التعامل مع المستخدم. ومن المرغوب فيه غالبا، إخفاء كامل القائمة عند حجب جميع بنودها. ولإخفاء بند فى قائمة باستخدام مصمم القوائم، نختار البند ثم نضبط خاصية Visible فى نافذة Properties على القيمة False . ولإخفاء بند القائمة باستخدام الكود، نستخدم الكود التالى فى أحد الوسائل الموجودة على النموذج:

```
MenuItem3.Visible = False
```

حذف بنود من القوائم

يعنى حذف أحد البنود من قائمة حذفة من مجموعة MenuItems فى كائن MainMenu المتعلق به. وقد تتطلب احتياجات وقت التشغيل استخدام البند بعد الاستغناء عنة، ولهذا يكون من المفضل إخفاء البند بدلا من حذفة. لحذف بند من قائمة فى وقت التصميم، ننقر على بند القائمة المستهدف فى مصمم القوائم، ثم نختار Delete. ولحذف أحد البنود باستخدام الكود، نستدعى وسيلة Remove فى تصنيف Menu.MenuItemCollection. المثال التالى، يقوم بتكوين هيكل قائمة يحتوى على بند mnuEuropeNations فى القمة. ويحتوى بند القمة المذكور على ثلاثة بنود: Italy، Germany، France. ويقوم الكود بحذف بند France من القائمة:

```
Private mnuCountries As MainMenu
Private mnuItemEuropeNations As MenuItem
Private mnuItemItaly As MenuItem
Private mnuItemGermany As MenuItem
Private mnuItemFrance As MenuItem


Public Sub CreateNationMenu ()
  mnuCountries = New MainMenu ()
  mnuItemEuropeNations = New MenuItem ("Some Nations of E&urope")
  mnuItemItaly = New MenuItem("&Italy")
  mnuItemGermany = New MenuItem("&Germany")
  mnuItemFrance = New MenuItem("&France")
  MnuCountries.MenuItems.Add (mnuItemEuropeNations)
  mnuItemEuropeNations.MenuItems.Add (mnuItemItaly)
  mnuItemEuropeNations.MenuItems.Add (mnuItemGermany)
  mnuItemEuropeNations.MenuItems.Add (mnuItemFrance)
        Me.Menu = mnuCountries
End Sub
  Public Sub RemoveFrance ()
  mnuItemEuropeNations.MenuItems.Remove (mnuItemFrance)
End Sub
```

دمج بنود القوائم

قد يكون من المفيد أحيانا إظهار بنود قائمتين معا فى قائمة واحدة، مثل الدمج بين قائمة حاوية ذات واجهة الاستخدام متعددة الوثائق مع القائمة التى على النموذج التابع النشط.

لإيضاح الكود اللازم لتحقيق عملية الدمج، نستدعى وسيلة MergMenu داخل أحد الإجراءات لكى نضيف كائنات MenuItem الخاصة بالقائمة التى نستدعيها إلى كائنات MenuItem بالقائمة التى تقوم بالاستدعاء. فى المثال التالى، نفترض وجود اثنين من مكونات MainMenu، هما : mnuCars، mnuTrucks.

```
Public Sub PutThemTogether ()
    mnuCars.MergeMenu (mnuTrucks)
End Sub
```

وهناك اثنان من الخصائص هما: MergeType و MergeOrder، تحددان كيفية التعامل مع البنود المفردة فى قائمة أثناء عملية الدمج والموقع النسبى لكل MenuItem فى القائمة الجديدة الناتجة عن الدمج. يمكن ضبط هذه الخصائص على انفراد أو مجتمعة لتحديد كيفية عرض البنود ومواقعها داخل القائمة الجديدة. المثال التالى يضبط الخصائص المذكورة بالنسبة لبند فى قائمة يدعى MenuItem1:

```
Public Sub SetMenuMergeProps ()
  MenuItem1.MergeType = MenuMerge.Add
  MenuItem1.MergeOrder = 1
End Sub
```

تعديل أسماء بنود قوائم نماذج الويندوز

عند استخدام مصمم القوائم، يتم ديناميكيا تكوين بنود القائمة، كما يتم ضبط خاصية Text بواسطة المستخدم. ويقوم النظام بتحديد أسماء البنود ديناميكيا باستخدام أرقام تمثل ترتيب تكوين البنود المختلفة. ويوفر Visual Studio محررا للأسماء داخل مصمم القوائم يتيح لنا التغيير السريع والسهل لخاصية Name. لتعديل أسماء بنود قائمة فى وقت تصميم التطبيق باستخدام محرر الاسم، نتبع الخطوات التالية:

١. فى مصمم القوائم، ننقر بزر الماوس الأيمن ونختار Edit Names. يؤدى ذلك إلى التحول إلى نمط تحرير الأسماء وظهور MenuItem Name إلى جانب النص المحدد فى خاصية Text.

٢. نغير خاصية Name إلى القيمة المرغوب بها. وللخروج من محرر الأسماء، ننقر بزر الماوس الأيمن مرة أخرى على داخل مصمم القوائم، ثم نختار Edit Names مرة أخرى.

تكوين قائمة لتتبع النماذج التابعة المفتوحة فى النموذج متعدد الوثائق

من بين خيارات تخطيط نماذج الويندوز، استخدام الواجهة متعددة الوثائق (MDI). ويمكن تتبع النوافذ التابعة التى يقوم النموذج بفتحها عن طريق استخدام خاصية MDIList بإحدى القوائم، كما يتضح من المثال التالى:

١. فى نافذة Properties، نقوم بضبط خاصية IsMDIContainer فى نموذج MDI على القيمة True.

٢. فى مربع Solution Explorer، ننقر بالزر الأيمن على المشروع، نشير إلى Add ثم ننقر على Windows Form.

٣. من مربع ToolBox، نسحب مكون MainMenu إلى سطح نموذج MDI الأصلى.

٤. نختار نموذج MDI الأصلى بالنقر علية. وفى نافذة Properties، نضبط خاصية Menu على قيمة MainMenu1.

٥. نضيف البنود التالية إلى القائمة الرئيسية فى مكون MainMenu باستخدام مصمم القوائم:

بند القائمة	النص
Menu Item	Text
MenuItem1	&File
MenuItem2	&Window

٦. فى مصمم القوائم، نضيف بند MenuItem3 إلى MenuItem1. نضبط خاصية Text على القيمة &New.

٧. فى نافذة الخصائص، نختار بند MenuItem2 من القائمة المنسدلة، ونضبط خاصية MDIList على قيمة True.

٨. نكون إجراء معالجة حدث Click للبند MenuItem3. داخل إجراء المعالجة، نضيف الكود التالى:

```
Protected Sub New_OnClick (ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MenuItem3.Click
  Dim NewMDIChild As New Form2 ()
  NewMDIChild.MDIParent = Me
```

```
  NewMDIChild.Show ()
End Sub
```

٩. نضغط مفتاح F5 لتشغيل التطبيق. نختار New من قائمة File لإنشاء نموذج MDI تابع.

١٠. ننقر قائمة Window فى النموذج الأصلى. سوف نلاحظ أن هذه القائمة تعرض أسماء جميع النماذج التابعة المفتوحة فى التطبيق.

تطوير قوائم الويندوز

هناك أربعة تحسينات يمكن إضافتها إلى القوائم لنقل المعلومات إلى المستخدمين:

- يمكن استخدام علامات الفحص لتحديد فتح وإقفال بعض السمات.
- يمكن استخدام مفاتيح الاختصار (Short Keys) للوصول إلى بنود قائمة أوامر داخل نموذج
- يمكن استخدام مفاتيح الوصول (Access Keys) للتنقل داخل القوائم باستخدام لوحة المفاتيح.
- يمكن استخدام شرائط الفصل (Separator bars) لوضع الأوامر ذات العلاقات فى مجموعات منفصلة داخل إحدى

لإضافة علامة فحص إلى قائمة فى وقت التصميم:

- نختار بند القائمة فى مصمم القوائم، ننقر المنطقة التى على يسار البند. يترتب على ذلك، ظهور علامة فحص تشير إلى أن الخاصية المشار إليها قد تم ضبطها على القيمة True.
- أو نختار بند القائمة فى مصمم القوائم ثم نضبط خاصية Checked على القيمة True فى مربع Properties. وبدلا من ذلك، يمكن ضبط خاصية RadioCheck على القيمة True، مما يؤدى إلى ظهور نقطة صغيرة على يسار البند عند اختياره.

لإضافة علامة فحص إلى بند قائمة باستخدام الكود:

- فى داخل الإجراء المستخدم لضبط خاصية بند القائمة، نضيف الكود التالى لضبط خاصية Checked على القيمة True:

```
myMnuItem.Checked = True
```

لإضافة مفتاح اختصار إلى بند قائمة فى وقت التصميم:

- نختار بند القائمة داخل مصمم القوائم.
- فى نافذة Properties، نضبط خاصية Shortcut على واحدة من القيم الموجودة فى القائمة المنسدلة.

لإضافة مفتاح اختصار إلى بند قائمة باستخدام الكود:

- داخل إجراء ضبط خصائص بند القائمة، نضيف كود لضبط خاصية Shortcut على واحدة من القيم الموجودة فى تعداد Shortcut:

```
To add a shortcut key to a menu item programmatically
myMnuItem.Shortcut = System.Windows.Forms.Shortcut.F6
```

لإضافة مفتاح وصول إلى بند قائمة:

عند ضبط خاصية Text، نضع رمز قبل الحرف الذى نريد وضع خط أسفله بصفته مفتاح وصول. على سبيل المثال، إدخال F&ormat فى خاصية Text ببند قائمة، سوف ينتج عنة ظهور هذا البند فى صيغة Format. وللانتقال إلى بند القائمة المذكورة، نضغط على مفتاح ALT لنقل التركيز إلى شريط القائمة، ونضغط المفتاح المحدد فى اسم البند لتنفيذه. وعندما تكون القائمة مفتوحة، يمكننا الضغط على مفتاح الوصول فقط بدون مفتاح ALT.

ولإضافة شريط فصل إلى قائمة:

- فى مصمم القوائم، ننقر بزر الماوس الأيمن على الموقع الذى نريد وضع شريط الفصل به ثم نختار New Separator.
- أو نضبط قيمة خاصية Text على القيمة (-) لجعل البند فاصل.

القوائم المختصرة

تستخدم القوائم المختصرة (Context Menus) داخل التطبيقات لتوفر للمستخدم إمكانية الوصول إلى الأوامر المتكررة باستخدام زر الماوس الأيمن. ويتم فى الغالب تخصيص هذا النوع من القوائم لأدوات التحكم لتوفير قوائم أوامر خاصة بالمتحكم.

إضافة القوائم المختصرة إلى نماذج الويندوز

يمكن إضافة قائمة مختصرة إلى نموذج ويندوز فى وقت التصميم بتنفيذ الخطوات التالية:

١. نفتح النموذج الذى نريد إضافة القائمة إلية فى مصمم نماذج الويندوز.

٢. فى مربع ToolBox، ننقر نقرا مزدوجا على مكون ContextMenu لإضافة قائمة إلى النموذج وإضافة مكون ContextMenu إلى مربع المكونات أسفل المصمم.

٣. نربط بين القائمة المختصرة وبين النموذج أو متحكم على النموذج عن طريق ضبط خاصية ContextMenu فى نافذة Properties.

لإضافة قائمة مختصرة إلى نموذج ويندوز باستخدام الكود:

١. فى محرر الكود، نكون إجراء جديد يشتمل على الكود اللازم لإضافة القائمة إلى النموذج.

٢. نضيف كود إلى الوسيلة مماثل للكود التالى لتكوين مثل من مكون ContextMenu:

```
Dim mnuContextMenu as New ContextMenu()
Me.ContextMenu = mnuContextMenu
```

بمجرد إضافة مكون ContextMenu إلى نموذج ويندوز، نحتاج إلى إضافة بنود قائمة إلية. يمكن استخدام مصمم القوائم لإضافة مثل هذه البنود فى وقت التصميم. ويتم الاحتفاظ ببنود القائمة فى مجموعة لكى يمكن إضافة بنود جديدة إلى القائمة المختصرة وقت التشغيل عن طريق إضافة كائنات MenuItem إلى المجموعة.

لإضافة بنود إلى القائمة المختصرة وقت التصميم:

١. ننقر مكون ContextMenu على نموذج الويندوز، مما يؤدى إلى ظهور نص "Type Here".

٢. ننقر النص السابق ونطبع اسم البند الجديد.

٣. لإضافة بند قائمة أخر، ننقر منطقة نص "Type Here" أخرى داخل مصمم القوائم على الترتيب التالى:

أ. ننقر منطقة على يمين بند القائمة الحالى لإضافة قائمة فرعية.

ب. ننقر منطقة أسفل البند الحالى لإضافة بند قائمة أخر.

لإضافة بنود قائمة إلى قائمة مختصرة باستخدام الكود:

١. نضيف مكون ContextMenu و نربطة مع نموذج ويندوز أو مع أداة تحكم على نموذج الويندوز بوضع الكود التالى فى إجراء عام داخل النموذج:

```
Public Sub AddContextMenuAndItems ()
  Dim mnuContextMenu as New ContextMenu()
  Me.ContextMenu = mnuContextMenu
End Sub
```

٢. داخل الإجراء السابق، نكون كائنات MenuItem لإضافتها إلى مجموعة كائنات ContextMenu باستخدام الكود التالى:

```
Dim mnuItemNew as New MenuItem ()
Dim mnuItemOpen as New MenuItem ()
```

٣. فى داخل الإجراء، نضبط خاصية Text لكل بند من بنود القائمة باستخدام الكود التالى:

```
mnuItemNew.Text = "&New"
mnuItemOpen.Text = "&Open"
```

٤. نضيف بنود القائمة إلى مجموعة باستخدام الكود التالى:

```
mnuContextMenu.MenuItems.Add (mnuItemNew)
mnuContextMenu.MenuItems.Add (mnuItemOpen)
```

ويمكن إضافة قائمة فرعية إلى البند الثانى بالقائمة، باستخدام الكود التالى:

```
Dim mnuItemOpenWith as new MenuItem ()
mnuItemOpenWith.Text = "Open &With..."
mnuItemOpen.MenuItems.Add (mnuItemOpenWith)
```

حذف بنود القائمة المختصرة

يمكن حذف البنود من القائمة المختصرة أثناء التصميم وأيضا باستخدام الكود. لحذف بنود من القائمة المختصرة فى وقت التصميم، نطبق الخطوات التالية:

١. نختار مكون ContextMenu على سطح النموذج.

٢. ننقر على نص البند لتركيز الضوء علية، ثم ننقر مرة أخرى لعرضة بلون مختلف

٣. نضغط مفتاح DELETE.

ولحذف بند من قائمة مختصرة باستخدام الكود:

١. نستخدم وسيلة RemoveAt فى خاصية MenuItems بمكون ContextMenu لحذف أحد البنود:

```
ContextMenu1.MenuItems.RemoveAt (0)
ContextMenu1.MenuItems.Remove (mnuItemNew)
```

لحذف جميع البنود من قائمة مختصرة:MenuItems فى خاصية Clearونستخدم وسيلة

```
ContextMenu1.MenuItems.Clear ()
```

الرسومات فى نماذج الويندوز

يطق على واجهة تنفيذ الرسومات فى Visual Basic .NET مصطلح +GDI. وتعتبر +GDI نسخة متطورة من واجهة GDI السابقة. وتسمح لنا +GDI بتكوين الرسومات، رسم النصوص، والتعامل معها على أنها كائنات. ولقد تم تصميم هذه الواجهة لكى تقدم مستوى مرتفع من الأداء وسهولة فى الاستخدام. ويمكن استخدام هذه الواجهة لتنفيذ الرسومات على نماذج الويندوز وأدوات التحكم. وقد حلت +GDI بالكامل محل GDI، وأصبحت هى الطريقة الوحيدة الآن لمعالجة الرسوم باستخدام الكود فى تطبيقات نماذج الويندوز.

أدوات الرسم باستخدام +GDI

قبل أن نستطيع رسم الخطوط والأشكال، تنفيذ النصوص، أو عرض ومعالجة الصور باستخدام +GDI، لابد من تنفيذ خطوتين أساسيتين:

١. تكوين كائن رسومات (Graphics Object). هذا الكائن يمكن تشبيهه بلوحة الرسم.
٢. استخدام كائن الرسومات فى رسم الخطوط والأشكال، كتابة النصوص، أو عرض ومعالجة الصور.

كائنات الرسم

يمكن تكوين كائن الرسم بعدة طرق:

- استقبال مرجع إلى كائن رسومات من خلال معامل PaintEventArgs فى حدث Paint.
- استدعاء وسيلة CreateGraphics فى متحكم أو نموذج للحصول على مرجع إلى كائن رسومات.
- تكوين كائن رسومات من أى كائن موروث من صورة.

معامل PaintEventArgs فى إجراء معالجة حدث Paint

عند برمجة إجراء معالجة حدث Paint فى أداة تحكم، يتم الحصول على كائن رسومات بصفته أحد المعاملات فى PaintEventArgs. للحصول على مرجع إلى كائن الرسومات من PaintEventArgs بحدث Paint:

١. نعلن عن كائن رسومات (Graphics Object).

٢. نخصص متغير للإشارة إلى الكائن الذى تم تمريرة فى PaintEventArgs.

٣. ندخل الكود اللازم لتلوين النموذج أو المتحكم.

الكود التالى، يوضح كيفية استخدام كائن الرسومات فى PaintEventArgs:

```
Private Sub Form1_Paint (sender As Object, pe As PaintEventArgs) Handles _
  MyBase.Paint
  Dim g As Graphics = pe.Graphics
End Sub
```

وسيلة CreateGraphics

يمكن استخدام وسيلة CreateGraphics فى أداة تحكم أو نموذج للحصول على مرجع إلى كائن رسومات يمثل سطح الرسم للنموذج أو الكائن. لاستخدام هذه الوسيلة، نتبع الخطوات التالية:

- نستدعى وسيلة CreateGraphics فى النموذج أو المتحكم التى نريد الرسم عليه، كما يتضح من الكود التالى:

```
Dim g as Graphics
g = Me.CreateGraphics
```

تكوين كائن رسومات من كائن صورة

بالإضافة إلى ما سبق، يمكن تكوين كائن رسومات من أى كائن مستنسخ من تصنيف Image. لتنفيذ ذلك، نستدعى وسيلة Graphics.FromImage، مع تزويدها باسم متغير الصورة التى نريد استخدامها فى تكوين كائن الرسومات (Graphics Object)، كما يتضح من الكود التالى:

```
Dim myBitmap as New Bitmap ("C:\myPic.bmp")
Dim g as Graphics = Graphics.FromImage (myBitmap)
```

رسم ومعالجة الصور والأشكال

بعد تكوين كائن Graphics، يمكن استخدامه فى رسم الخطوط والأشكال، تنفيذ النصوص، أو عرض ومعالجة الصور. والتصنيفات (Classes) الأساسية المستخدمة مع كائن Graphics لتنفيذ هذه المهام هى:

- تصنيف Pen الذى يستخدم فى رسم الخطوط، حدود الأشكال، أو الأشكال الهندسية الأخرى.
- تصنيف Brush الذى يستخدم فى تعبئة المناطق الرسومية، مثل الأشكال والصور بالألوان.
- تصنيف Font الذى يوفر مجموعات الحروف التى يمكن استخدامها عند كتابة النصوص.
- هيكل بيانات Color الذى يمثل الألوان المختلفة التى يمكن استخدامها.

الأقلام، الفرش، والألوان

نستخدم كائنات القلم والفرشاة لتنفيذ الرسومات، النصوص، والصور باستخدام واجهة GDI+. والقلم هو كائن من تصنيف Pen، ويستخدم كما سبق ايضاحة فى رسم الخطوط وحدود الأشكال. والفرشاة هى كائن من أى تصنيف موروث من تصنيف Brush، وتستخدم فى تعبئة الأشكال أو تلوين النصوص. كائنات Color هى عبارة عن أمثلة من تصنيفات تمثل لون معين، ويمكن أن تستخدم بواسطة الأقلام والفرش لتحديد لون الرسم الذى يتم تنفيذه.

الأقلام (Pens)

توضح الأمثلة التالية كيفية تكوين قلم أسود.

- تكوين قلم اسود بكثافة افتراضية قدرها واحد.

```
Dim myPen as New Pen (Color.Black)
```

- تكوين قلم اسود بكثافة قدرها ٥

```
Dim myPen as New Pen (Color.Black, 5)
```

- تكوين قلم من كائن Brush موجود

```
Dim myPen as New Pen (myBrush)
```

- تكوين قلم من كائن Brush وكثافة قدرها ٥

```
Dim myPen as New Pen (myBrush, 5)
```

- استخدام القلم لرسم مقطع

```
Dim myPen as New Pen (Color.Black)
Dim g as Graphics = Me.CreateGraphics
g.DrawEllipse (myPen, 20, 30, 10, 50)
```

وبمجرد تكوين القلم، يمكن تغيير الكثير من خصائصه التى تؤثر على الطريقة التى يعرض بها الخطوط. من هذه الخصائص، خاصية Width، خاصية Color التى تؤثر على مظهر الخط. خاصية StartCap وخاصية EndCap ، التى تمكننا من إضافة أشكال إلى بداية أو نهاية الخط.

الفرشاة

تعتبر الفرشاة كائن يستخدم مع كائن Graphics لتكوين الأشكال وكتابة النصوص. و هناك عدة أنماط من الفرش تشمل: SolidBrush، HatchBrush، TextureBrush، LinearGradientBrush، PathGrandientBrush، وجميعها ترث من تصنيف Brush.

يوضح المثال التالى، كيفية رسم مقطع ذات لون أحمر أصم على النموذج. ويتوافق المقطع مع حجم المستطيل المستخدم معه، وهو فى هذه الحالة كامل النموذج.

```
Dim g as Graphics = Me.CreateGraphics
Dim myBrush as New SolidBrush (Color.Red)
g.FillEllipse(myBrush, ClientRectangle)
```

الألوان

يمثل هيكل بيانات Color أنواع مختلفة من الألوان التى تستخدم مع الأقلام والفرش لتحديد اللون المستخدم. ويمكن تقسيم الألوان إلى قسمين: الألوان التى يحددها النظام (System-Defined Colors)، والألوان التى يحددها المستخدم (User Defined Colors). حيث يحتوى النظام على العديد من الألوان التى يمكن الوصول إليها من خلال هيكل الألوان (Color Structure). من أمثلة هذه الألوان:

```
Dim myColor as Color
myColor = Color.Red
myColor = Color.Aquamarine
myColor = Color.LightGoldenrodYellow
myColor = Color.PapayaWhip
myColor = Color.Tomato
```

ويمكن أيضا أن يقوم المستخدم بتكوين ألوان خاصة به باستخدام وسيلة

Color.FromArgb. تسمح لنا هذه الوسيلة بتحديد نسبة كل من اللون الأحمر، اللون الأزرق، واللون الأخضر التى تكون اللون. المثال التالى، يوضح استخدام هذه الوسيلة:

```
Dim myColor as Color
myColor = Color.FromArgb (23,56,78)
```

تنفيذ الرسم باستخدام GDI+

يحتوى كائن Graphics على وسائل لرسم أشكال متنوعة من الخطوط والأشكال. ويمكن رسم الأشكال البسيطة والمعقدة باستخدام الألوان الصماء و الشفافة، أو استخدام الألوان التى يقوم المستخدم بتحديدها. ويستخدم كائن Pen فى رسم الخطوط، المنحنيات، وحدود الأشكال. ولتعبئة منطقة بالألوان، مثل مستطيل أو دائرة، نستخدم الفرشاة.

رسم الخطوط والأشكال

لرسم أحد الخطوط أو حدود أحد الأشكال:

١. نحصل على مرجع إلى كائن Graphics الذى سوف نقوم باستخدامه فى الرسم.

```
Dim g as Graphics = Button1.CreateGraphics
```

٢. نكون مثلٌ من تصنيف Pen، الذى نريد استخدامه فى رسم الخطوط ونضبط الخصائص اللازمة.

```
Dim myPen as new Pen (Color.Red)
myPen.Width = 5
```

٣. نستدعى الوسيلة المناسبة للشكل المطلوب رسمة، وإدخال المعاملات المطلوبة، مثل:

Graphics.DrawLine، Graphics.DrawPolygon، Graphics.DrawRectangle.

```
g.DrawLine(myPen, 1, 1, 45, 65)
g.DrawBezier(myPen, 15, 15, 30, 30, 45, 30, 87, 20)
g.DrawEllipse(myPen, New Rectangle(33, 45, 40, 50))
g.DrawPolygon(myPen, New PointF() {New PointF(1, 1), _
   New PointF (20, 10), New PointF(5, 4), New PointF(100, 2), _
   New PointF(200, 50), New PointF(39, 45)})
```

ولرسم شكل غير فارغ:

١. نحصل على مرجع إلى كائن Graphics المستخدم فى الرسم.

```
Dim g as Graphics = Button1.CreateGraphics
```

٢. نكون مثل من الفرشاة التى نريد استخدامها لتلوين الشكل

```
Dim myBrush as New SolidBrush (Color.Red)
```

٣. نستدعى الوسيلة المناسبة للشكل الذى نريد تلوينه، مع إدخال المعاملات المطلوبة.

من أمثلة هذه الوسائل: FillPolygon، FillRectangle، FillPath.

```
g.FillPolygon(myBrush, New PointF() {New PointF(20, 20), _
   New PointF (50, 100), New PointF (60, 10), New PointF (200, 4), _
   New PointF (0, 0), New PointF (20, 20)})
g.FillRectangle(myBrush, New RectangleF(50, 50, 100, 100))
g.FillPie(myBrush, New Rectangle(110, 110, 300, 300), 0, 90)
```

رسم النصوص

يمكن استخدام كائن Graphics لتنفيذ النصوص. يتطلب ذلك وجود كائن Brush لتحديد نمط تعبئة النص، وكائن Font لتحديد النمط الذى سوف يتم تعبئته. لكتابة سلسلة من الرموز باستخدام +GDI، ننفذ الخطوات التالية:

١. نحصل على مرجع إلى كائن Graphics لإستخدامة فى الرسم.

```
Dim g as Graphics = Button1.CreateGraphics
```

٢. نكون مثل من الفرشاة التى سوف نستخدمها لتلوين النص.

```
Dim mybrush As New Drawing2D.LinearGradientBrush (ClientRectangle, _
   Color.Red, Color.Yellow, Drawing2D.LinearGradientMode.Horizontal)
```

٣. نكون نمط الحروف المطلوب استخدامه لعرض النص.

```
Dim myFont as New Font ("Times New Roman", 24)
```

٤. نستدعى وسيلة Graphics.DrawString لتنفيذ النص.

أ. عند استخدام مستطيل، سوف يلتف النص داخل هذا المستطيل.

ب. أو يبدأ عند الإحداثيات التى ندخلها.

```
g.DrawString ("Look at this text!", myFont, myBrush, New _
   RectangleF (10, 10, 100, 200))
g.DrawString ("Look at this text!", myFont, myBrush, 10, 10)
```

عرض الرسومات

يمكن استخدام +GDI لعرض الصور الموجودة فى ملفات بالتطبيقات. يتم ذلك عن طريق تكوين كائن من تصنيف Image، مثل Bitmap، تكوين كائن Graphics، واستخدام وسيلة DrawImage. يترتب على ذلك وضع الصور على سطح الرسومات الذى يمثله تصنيف Graphics. ويمكن استخدام محرر الرسومات (Image Editor) لتكوين وتعديل ملفات الرسومات فى وقت التصميم، وتنفيذ ذلك فى وقت التشغيل. لعرض صورة باستخدام +GDI، نتبع الخطوات التالية:

١. نكون كائن يمثل الصورة التى نريد عرضها. ويجب أن يكون هذا الكائن عضوا فى

تصنيف يرث من تصنيف Image، مثل تصنيف Bitmap، أو تصنيف MetaFile. كما يتضح من الكود التالى:

```
Dim myBitmap as New Bitmap ("D\TestImage.bmp")
```

٢. نكون كائن Graphics يمثل سطح الرسم المستخدم.

```
Dim g as Graphics = Me.CreateGraphics
```

٣. نستدعى وسيلة Graphics.DrawImage فى كائن Graphics لإنتاج الصورة. ويجب تحديد كلا من الصورة المطلوب رسمها، وإحداثيات الموقع المطلوب الرسم به، كما هو موضح فى الكود التالى:

```
g.DrawImage(myBitmap, 1, 1)
```

دعم الطباعة فى نماذج الويندوز

تعتمد الطباعة فى نماذج الويندوز على استخدام مكون PrintDocument لتمكين المستخدم من تنفيذ عملية الطباعة، متحكم PrintPreviewDialog، مكون PrintDialog، ومكون PageSetupDialog لتوفير واجهة الاستخدام الرسومية المألوفة أمام المستخدمين بهدف التوافق مع نظام تشغيل الويندوز. والطريقة القياسية للقيام بعملية الطباعة تشمل تكوين مثل من تصنيف PrintDocument، ضبط الخصائص التى تحدد ما سوف نقوم بطباعته باستخدام تصنيف PrinterSettings وتصنيف PageSettings، واستدعاء وسيلة Print لطباعة المستند.

تكوين وظائف الطباعة

أساس الطباعة فى نماذج الويندوز هو حدث PrintPage الموجود بمكون PrintDocument. وعن طريق كتابة كود لمعالجة هذا الحدث، يمكننا تحديد ماذا نطبع وكيف نطبع.

لتكوين وظيفة طباعة فى وقت التصميم، نتبع الخطوات التالية:

١. نضيف مكون PrintDocument إلى النموذج.

٢. ننقر بزر الماوس الأيمن على النموذج ونختار View Code.

٣. نكتب الكود اللازم لمعالجة حدث PrintPage. يشتمل هذا الكود على تحديد الأسلوب المنطقى المتبع فى عملية الطباعة، وتحديد المادة المطلوب طباعتها. فى

المثال التالى، يتم تكوين مستطيل أحمر اللون فى إجراء معالجة حدث PrintPage، لاستخدامه عينة للطباعة:

```
Private Sub PrintDocument1_PrintPage (ByVal sender As Object, ByVal e As System.Drawing.Printing.PrintPageEventArgs) Handles PrintDocument1.PrintPage
  e.Graphics.FillRectangle(Brushes.Red, New Rectangle(500, 500, 500, 500))
End Sub
```

تغيير خيارات الطباعة وقت التشغيل

هناك أوقات نحتاج فيها إلى تغيير خيارات الطباعة أثناء تشغيل التطبيق. ويرجع السبب فى ذلك غالبا إلى الخيارات التى يحددها المستخدم. لتغيير هذه الخيارات، نستخدم مكون PrintDialog وتصنيف PrinterSettings باتباع الخطوات التالية:

١. من مربع ToolBox، نضيف مكون PrintDialog إلى النموذج.

٢. نضبط خاصية Document على اسم المستند المطلوب طباعته.

٣. ننقر بزر الماوس الأيمن على النموذج ونختار View Code.

٤. نعرض مكون PrintDialog باستخدام وسيلة ShowDialog.

```
PrintDialog1.ShowDialog ()
```

٥. نحصل على خيارات الطباعة التى يحددها المستخدم عن طريق خاصية PrinterSettings فى مركب PrintDialog.

اختيار الطابعات

يحتاج المستخدم فى الغالب إلى استخدام طابعة غير الطابعة الافتراضية. ويتم اختيار طابعة من بين الطابعات المثبتة باستخدام مكون PrintDialog. ويجرى استخدام النتيجة العائدة من هذا المركب فى اختيار الطابعة. لاختيار طابعة وطباعة ملف، ننفذ الخطوات التالية:

١. نستخدم إجراء النقر على Button لإدخال الكود التالى الخاص بتكوين مثل من تصنيف PrintDialog والحصول على الطابعة التى يختارها المستخدم عن طريق خاصية DialogResult.

```
Private Sub Button1_Click (ByVal sender As Object, ByVal e As System.EventArgs) Handles Button1.Click
  Dim PrintDialog1 As New PrintDialog ()
```

```
  PrintDialog1.Document = PrintDocument1
  Dim result As DialogResult = PrintDialog1.ShowDialog ()

  If (result = DialogResult.OK) Then
    PrintDocument1.Print ()
  End If
End Sub
```

٢. نستخدم إجراء معالجة حدث PrintPage في مكون PrintDocument لإدخال الكود التالى الذى يقوم بتحديد الوثيقة المطلوب طباعتها على الطابعة السابق اختيارها.

```
Private Sub PrintDocument1_PrintPage (ByVal sender As Object, ByVal e As System.Drawing.Printing.PrintPageEventArgs) Handles PrintDocument1.PrintPage
  e.Graphics.FillRectangle(Brushes.Red, New Rectangle(500, 500, 500, 500))
End Sub
```

طباعة الرسوم والاشكال

يوفر لنا تصنيف Graphics الوسائل اللازمة لرسم الكائنات على أدوات مختلفة، مثل شاشة الكمبيوتر والطابعة. لطباعة الرسوم والأشكال، ننفذ الخطوات التالية:

١. نضيف مكون PrintDocument إلى النموذج.

٢. ننقر بزر الماوس الأيمن على النموذج ونختار View Code.

٣. في إجراء معالجة حدث PrintPage، نرسل تعليمات الرسم الذى يجب طباعته إلى الطابعة باستخدام خاصية Graphics في تصنيف PrintPageEventArgs.

لتوضيح ذلك، نستخدم إجراء معالجة حدث PrintPage في المثال التالى لتكوين رسم هندسي ازرق اللون داخل حدود مستطيل:

```
Private Sub PrintDocument1_PrintPage (ByVal sender As Object, ByVal e As System.Drawing.Printing.PrintPageEventArgs) Handles PrintDocument1.PrintPage
  e.Graphics.FillEllipse(Brushes.Blue, New Rectangle(100, 150, 250, 250))
End Sub
```

طباعة النصوص

تعتبر طباعة النصوص من الوظائف الشائعة جدا في تطبيقات الويندوز. لتنفيذ هذه المهمة، نتبع الخطوات التالية:

١. نضيف مكون PrintDocument إلى النموذج.

٢. ننقر بزر الماوس الأيمن على النموذج نختار View Code.

٣. فـى إجـراء معالجـة حـدث PrintPage، نحـدد الـنص المطلـوب طباعته باستخدام خاصـية Graphics فـى تصـنيف PrintPageEventArgs. لإيضـاح ذلـك، نسـتخدم إجـراء معالجـة حـدث PrintPage لطـباعة نـص "Sample Text" بـاللون الأسـود باستخدام طاقم حروف Arial:

```
Private Sub PrintDocument1_PrintPage (ByVal sender As Object, ByVal e As
System.Drawing.Printing.PrintPageEventArgs) Handles PrintDocument1.PrintPage
  e.Graphics.DrawString("SampleText", New Font("Arial", 80, FontStyle.Bold),
Brushes.Black, 150, 125)
End Sub
```

استكمال وظائف الطباعة

من المعتاد أن تقوم تطبيقات معالجة الكلمات وغيرها من التطبيقات التى تتعامل مع وظائف الطباعة بتوفير خيار عرض رسالة توضح أن وظيفة الطباعة قد تم الانتهاء منها. يتم ذلك فى نماذج الويندوز عن طريق معالجة حدث EndPrint فى مكون PrintDocument.

يفترض الإجراء التالى أننا قمنا بتكوين تطبيق ويندوز يحتوى على مكون PrintDocument. وهو النمط القياسى للطباعة فى تطبيقات الويندوز. لتوضيح استكمال وظيفة الطباعة، ننفذ الخطوات التالية:

١. فى نافذة Properties، نضبط خاصية DocumentName بمكون PrintDocument. وبدلا من ذلك، يمكن ضبط هذه الخاصية باستخدام الكود التالى:

```
PrintDocument1.DocumentName = "MyTextFile"
```

٢. ننقر بزر الماوس الأيمن على النموذج ونختار View Code.

٣. فى إجراء معالجة حدث EndPrint، ندخل الكود التالى الذى يقوم بعرض مربع رسالة تشير إلى أن المستند قد تم الانتهاء من طباعته.

```
Private Sub PrintDocument1_EndPrint (ByVal sender As Object, ByVal e As
System.Drawing.Printing.PrintEventArgs) Handles PrintDocument1.EndPrint
  MessageBox.Show (PrintDocument1.DocumentName + " has finished printing.")
End Sub
```

الطباعة المبدئية

تعتبر الطباعة المبدئية من السمات الشائعة فى تطبيقات الويندوز. حيث يتم عرض

نموذج من الطباعة المبدئية للمستند المطلوب طباعته على شاشة الكمبيوتر أمام المستخدم. وتوفر نماذج الويندوز هذه الخاصية من خلال استخدام متحكم PrintPreviewDialog. لتنفيذ عملية الطباعة المبدئية، نستخدم الخطوات التالية:

١. نضيف متحكم PrintPreviewDialog إلى النموذج من مربع ToolBox.

٢. نضبط خاصية Document في ذلك المتحكم على اسم مكون PrintDocument، الذى نستخدمه فى طباعة المستند.

٣. ننقر بزر الماوس الأيمن على النموذج ونختار View Code.

٤. نعرض متحكم PrintPreviewDialog باستخدام وسيلة ShowDialog.

الكود التالى يعرض متحكم PrintPreviewDialog نتيجة اختيار بند فى قائمة miPrintPrev:

```
Private Sub miPrintPrev_Click (ByVal sender As Object, ByVal e As System.Drawing.Printing.PrintPageEventArgs) Handles miPrintPrev.Click
  PrintPreviewDialog1.ShowDialog ()
End Sub
```

عمليات السحب والإسقاط ولوحة القص

من العمليات المألوفة لدى المستخدمين، سحب الكائنات من منطقة على شاشة الكمبيوتر إلى أخرى. ويتم تمكين المستخدم من عملية السحب والإسقاط (Drag and Drop) داخل تطبيقات الويندوز من خلال معالجة سلسلة من الأحداث، مثل DragLeave، DragEnter، و DragDrop. وعن طريق استخدام المعلومات المتاحة من خلال معاملات هذه الأحداث، يمكن تسهيل عمليات السحب والإسقاط أمام المستخدمين. كما أن هناك العديد من التطبيقات التى تستخدم لوحة القص (Clipboard) بصفة سجل مؤقت للبيانات. على سبيل المثال، تستخدم تطبيقات معالجة الكلمات لوحة القص أثناء عمليات القص، النسخ، والإسقاط.

تنفيذ عمليات القص واللصق فى نماذج الويندوز

تبدأ عمليات السحب والإسقاط فى تطبيقات الويندوز بعملية السحب وتنتهي بعملية الإسقاط. وتستخدم إجراءات لمعالجة سلسلة من الأحداث، أهمها DragDrop، DragLeave، DragEnter.

سحب البيانات

يوضح المثال التالى، استخدام حدث MouseDown لبدء عملية السحب. ويمكن استخدام إجراء معالجة أى حدث للقيام بهذه المهمة. فى الكود التالى، نستخدم وسيلة DoDragDrop فى إجراء معالجة MouseDown بأحد الأزرار للقيام بسحب سلسلة من الرموز:

```
Private Sub Button1_MouseDown (ByVal sender As Object, ByVal e As System.Windows.Forms.MouseEventArgs) Handles Button1.MouseDown
  Button1.DoDragDrop (Button1.Text, DragDropEffects.Copy Or DragDropEffects.Move)
End Sub
```

إسقاط البيانات

بمجرد بدء عملية السحب من موقع على نموذج ويندوز أو متحكم، من الطبيعى أن تنتهى العملية بإسقاط البيانات فى مكان ما. وسوف يتغير شكل مؤشر الماوس عند المرور بمنطقة مؤهلة للإسقاط بنموذج الويندوز. ويمكن جعل أى منطقة بالنموذج أو المتحكم مؤهلة لقبول إسقاط البيانات عن طريق ضبط خاصية AllowDrop ومعالجة أحداث DragEnter و DragDrop.

لتنفيذ إسقاط البيانات، ننفذ الخطوات التالية:

١. فى نافذة Properties، نضبط خاصية AllowDrop على القيمة True.

٢. ننقر بزر الماوس الأيمن على النموذج فى Solution Explorer ونختار View Code.

٣. فى إجراء معالجة DragEnter بالمتحكم الذى سوف يتم به الإسقاط، نستخدم عبارة If للتحقق من نوع البيانات التى يتم إسقاطها، كما يتضح من الكود التالى:

```
Private Sub TextBox1_DragEnter (ByVal sender As Object, ByVal e As System.Windows.Forms.DragEventArgs) Handles TextBox1.DragEnter
  If (e.Data.GetDataPresent (DataFormats.Text)) Then
    e.Effect = DragDropEffects.Copy
  Else
    e.Effect = DragDropEffects.None
  End If
End Sub
```

٤. فى إجراء معالجة حدث DragDrop الخاص بالمتحكم الذى سوف يتم به الإسقاط،

نستخدم وسيلة GetData لاستخراج البيانات التى يتم سحبها، كما يتضح من الكود التالى:

```
Private Sub TextBox1_DragDrop (ByVal sender As Object, ByVal e As System.Windows.Forms.DragEventArgs) Handles TextBox1.DragDrop
  TextBox1.Text = e.Data.GetData (DataFormats.Text).ToString
End Sub
```

وضع البيانات على لوحة القص

هناك عدد من التطبيقات التى تستخدم لوحة (Clipboard) للتسجيل المؤقت للبيانات. على سبيل المثال، يستخدم تطبيق معالجة الكلمات هذه الخاصية أثناء عمليات القطع/النسخ/اللصق (Cut/Copy/Paste). وتشتمل عمليات لوحة القص على تسجيل البيانات بلوحة القص واسترجاع البيانات من لوحة القص.

وضع البيانات فى لوحة القص

نستخدم وسيلة SetDataObject لإرسال البيانات إلى لوحة القص (Clipboard). المثال التالى، يبين كيفية استخدام حدث Click فى متحكم Button لإرسال النص الموجود فى مربع نص إلى لوحة القص:

```
Private Sub Button1_Click (ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
  Clipboard.SetDataObject (TextBox1.Text)
End Sub
```

استرجاع البيانات من لوحة القص

لاستخراج البيانات المخزنة فى لوحة القص، ننفذ الخطوات التالية:

١. نسحب أدوات التحكم Button و TextBox من مربع ToolBox إلى النموذج.

٢. ننقر نقرا مزدوجا على متحكم Button لتكوين إجراء معالجة حدث Click، مما يترتب علية فتح محرر الكود ووضع نقطة الإدراج داخلة.

٣. فى إجراء معالجة حدث Click، نكتب الكود اللازم للحصول على البيانات من لوحة القص بتكوين مثل من الواجهة البينية لكائن البيانات (IDataObject) ثم الحصول على البيانات من لوحة القص باستخدام وسيلة GetDataObject. نختبر صلاحية البيانات الموجودة باستخدام وسيلة GetDataPresent. وعندما يكون نوع

البيانات صحيحا، نستخدم خاصية GetData للحصول على البيانات ووضعها فى خاصية Text بمربع النص:

```
Private Sub Button1_Click (ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
  Dim data As IDataObject = Clipboard.GetDataObject ()
  If (data.GetDataPresent (DataFormats.Text)) Then
    TextBox1.Text = data.GetData (DataFormats.Text).ToString()
  End If
End Sub
```

المساعدة فى نماذج الويندوز

يعتبر نظام المساعدة جزءا ضروريا فى بناء تطبيقات الويندوز، حيث يلجأ المستخدمون إلية طلبا للعون عند مواجهة المشاكل وعدم الوضوح. وتدعم نماذج الويندوز نوعين من المساعدة، كل منهما يتم الوصول إلية من خلال مكون HelpProvider. النوع الأول، يشتمل على توجيه المستخدم إلى ملف مساعدة (Help File) يحتوى على نصوص بصيغة HTML أو HTMLHelp. والنوع الثانى يعرض مساعدة طافية مختصرة متعلقة بأدوات التحكم المختلفة. ويعتبر ذلك مفيدا بصفة خاصة فى مربعات الحوار. ويمكن استخدام كلا الأسلوبين على نفس النموذج. ويوجد أسلوب أخر للمساعدة يتمثل فى استخدام الإيضاحات المختصرة (ToolTips) لتقديم نبذة خاصة بكل أداة من بأدوات التحكم على نماذج الويندوز.

ملفات المساعدة

يمكن تقديم المساعدة من خلال وضع موضوعات المساعدة المختلفة فى صيغة HTML و HTMLHelp داخل ملفات. وهناك العديد من التطبيقات التى تمكننا من تكوين هذه الملفات، مثل HTML Help Workshop. يلى ذلك ربط موضوعات المساعدة مع أدوات تحكم محددة داخل نماذج الويندوز من خلال مكون HelpProvider. ولإيضاح كيفية تقديم هذا النوع من المساعدة، ننفذ الخطوات التالية:

١. نسحب مكون HelpProveder من مربع ToolBox إلى سطح النموذج.

٢. فى نافذة Properties، نضبط خاصية HelpNamespace على ملف مساعدة من نوع col، chm، أو htm.

٣. نختار أحد أدوات التحكم على النموذج ثم نضبط خاصية HelpKeyword فى نافذة

Properties. تحتوى هذه الخاصية على سلسلة الحروف التى يتم تمريرها من خلال مكون HelpProvider إلى ملف مساعدة لاستدعاء موضوع المساعدة.

٤. فى نافذة Properties، نضبط خاصية HelpNavigator على قيمة فى تعداد HelpNavigator لتحديد أسلوب تمرير قيمة خاصية HelpKeyword إلى نظام المساعدة. الجدول رقم (١٧) يبين قيم الضبط الممكنة لهذة الخاصية وإيضاحاتها.

اسم العضو	الإيضاح
AssociateIndex	تحدد تنفيذ فهرس موضوع معين فى عنوان URL محدد
Find	تعرض صفحة البحث فى عنوان URL محددة
Index	تعرض فهرس عنوان URL معين
KeywordIndex	تحدد كلمة مرشدة للبحث وتنفيذ فعل فى عنوان URL محدد
TableOfContents	تعرض جدول محتويات ملف المساعدة
Topic	تعرض الموضوع الذى يشير إلية عنوان URL المحدد.

جدول ١٧

عند الضغط فى وقت التشغيل على F1 عندما يكون المتحكم السابق ضبط خاصية HelpKeyword و خاصية HelpNavigator به فى بؤرة التركيز، يتم فتح ملف المساعدة المرتبط مع مكون HelpProvider.

المساعدة الطافية

يمكن تقديم المساعدة على نماذج الويندوز من خلال أزرار المساعدة الموجودة على الجانب الأيمن من شريط العنوان فى النموذج، والتى يمكن الوصول إليها من خلال خاصية HelpButton. ويعتبر تقديم المساعدة بهذه الطريقة مناسبا للاستخدام مع مربعات الحوار لأن مربعات الحوار تواجه صعوبات عند التعامل مع ملفات المساعدات، كما أن استخدام زر المساعدة يتطلب عدم وجود أزرار التصغير والتكبير فى شريط العنوان.

لعرض المساعدة الطافية، ننفذ الخطوات التالية:

١. نسحب مكون HelpProvider من مربع ToolBox إلى النموذج.
٢. في نافذة Properties، نضبط خاصية HelpButton على القيمة True. يترتب على ذلك، ظهور زر يحتوى على علامة الاستفهام بالجانب الأيمن من شريط العنوان (Tool Bar). لكى يمكن عرض زر Help Button، يجب التحقق من ضبط خصائص Minimize وMaximize على القيمة False.
٣. على النموذج، نختار أداة التحكم التى نريد عرض المساعدة لها ثم نضبط خاصية HelpString في نافذة Properties.
٤. نضغط F5 لتشغيل التطبيق.
٥. نضغط زر المساعدة على شريط العنوان ثم ننقر على المتحكم الذى سبق ضبط خاصية HelpString له.

المساعدة المختصرة

يستخدم مكون ToolTip لعرض إيضاحات مختصرة خاصة بأحد أدوات التحكم على نموذج من نماذج الويندوز. ويحتوى هذا المكون على خاصية تحدد النص الذى يتم عرضة لكل متحكم على النموذج. لإيضاح تنفيذ إيضاحات ToolTip، ننفذ الخطوات التالية:

١. نسحب متحكم ToolTip من مربع ToolBox إلى سطح النموذج.
٢. نختار أداة تحكم على سطح النموذج ثم نضبط خاصية ToopTip على النص الذى نريد عرضة.
٣. نضغط F5 لتشغيل التطبيق ثم نضع مؤشر الماوس على أداة التحكم السابق اختيارها برهة قصيرة لعرض الإيضاح المختصر.

استخدام الثقافة العربية فى نماذج الويندوز

يمكن تعريف الثقافة بأنها اللغة المستخدمة وما يرتبط بها من قيم مثل طريقة الكتابة، التقويم، الأرقام، العملة، وغيرها من المعلومات. ويجب تخطيط تطبيقات الكمبيوتر بالطريقة التى تجعلها صالحة للعمل فى ظل ثقافات مختلفة لكى تكون قابلة للتوزيع عالميا، وهو ما يطلق علية العولمة (Globalization). كما يجب توفير إمكانية ترجمة واجهات التعامل مع

المستخدمين إلى ثقافاتهم المحلية، وهو ما يطلق علية الأقلمة (Localization). كما يجب توفير الأدوات اللازمة لتحويل إدخالات لوحة المفاتيح إلى اللغة الخاصة بالثقافة المستخدمة، وهو ما يطلق علية لغة الإدخال (Input Language). وتعتبر الثقافة العربية من الثقافات الأساسية فى عالم اليوم، ولهذا يجب العناية بتوفير التطبيقات التى تستخدمها، سواء فى واجهات التعامل أو فى إدخال البيانات.

أقلمة وعولمة تطبيقات الويندوز

يشتمل نظام NET Framework على الكثير من الخدمات المبنية التى تجعل من السهل تطوير تطبيقات الويندوز التى تتوافق مع ثقافات محلية مختلفة. ولقد تم تصميم Visual Studio .NET من البداية أيضا لتسهيل تطوير التطبيقات التى تستهدف الجمهور العالمي عن طريق الاستفادة من الخدمات المبنية فى نظام NET Framework.

عولمة تطبيقات الويندوز

تعنى العولمة تصميم التطبيقات لكى تتوافق مع ثقافات مختلفة. ولتحقيق هذا الغرض، يتكون نموذج NET Localization من وحدة تجميع (Assembly) أساسية تحتوى على كود التطبيق والموارد التى يعتمد عليها، مثل السلاسل، الرسومات، والكائنات الأخرى، باللغة المستخدمة فى تطوير التطبيق. بالإضافة إلى ذلك، يحتوى التطبيق على وحدات تجميع خاصة بالثقافات الأخرى التى يتعامل معها التطبيق. تحتوى كل وحدة تجميع من هذه الوحدات على الموارد الخاصة بالثقافة المحلية التى تخص الوحدة. ويسمح نظام مشروعات تطبيقات نماذج الويندوز ببناء ملفات موارد واجه الاستخدام الأساسية ووضعها فى وحدة التجميع الأساسية للتطبيق، وبناء ملفات موارد خاصة بواجهات استخدام الثقافات الأخرى ووضعها فى وحدات تجميع إضافية خاصة بتلك الثقافات.

ضبط الثقافة المستخدمة

يستخدم Visual Basic .NET قيمتان من قيم الثقافة: ثقافة واجهة الاستخدام (UI Culture)، والثقافة المستخدمة (Current Culture). القيمة الأولى تحدد الموارد التى يتم تحميلها عند تشغيل التطبيق. والقيمة الثانية تحدد طريقة صياغة قيم، مثل العملة، التاريخ، والأرقام. الخاصية التى تتعامل مع القيمة الأولى هى خاصية CurrentUICulture، والخاصية التى تتعامل مع القيمة الثانية هى خاصية CurrentCulture. ويمكن ضبط هاتين

الخاصتين عند الرغبة فى استخدام ضوابط ثقافية غير الموجودة بنظام التشغيل، باستخدام الكود التالى:

- نضع العبارتين التاليتين فى بداية وحدة الكود.

```
Imports System.Threading
Imports System.Globalization
```

- نضع الكود التالى قبل وسيلة InitializeComponent.

```
Thread.CurrentThread.CurrentCulture = new CultureInfo ("ar-SA")
Thread.CurrentThead.CurrentUICulture = new CultureInfo ("ar-SA")
```

- بالنسبة لسلاسل الرموز التى يجب أن تبقى ثابتة بغض النظر عن الثقافة، نستدعى وسائل الصياغة مع استخدام معامل CultureInfo.InvariantCulture، كما يتضح من المثال التالى:

```
Dim MyInt As Integer = 100
Dim MyString As String = MyInt.ToString ("C", CultureInfo.InvariantCulture)
MessageBox.Show (MyString)
```

عرض النصوص من اليمين إلى اليسار

بعض اللغات، مثل اللغة العربية، يتم كتابتها من اليمين إلى اليسار. كما يرتبط استخدام هذه الثقافات أيضا بتغيير تصفيف أدوات التحكم الموجودة على النماذج من اليمين إلى اليسار. والخاصية التى تتحكم فى طريقة العرض المذكورة هى خاصية RightToLeft بنماذج الويندوز وأدوات التحكم التى توضع عليها. عند القيام بضبط خاصية RightToLeft على القيمة Yes، يتم عكس اتجاه الكتابة من اليمين إلى اليسار وعكس تصفيف الكائنات الموجودة على النموذج، مثل مربعات الاختيار (Check Boxes)، شرائط التدرج(Scroll Bars) الأفقية والرأسية، النصوص الموجودة فى شرائط العناوين.

أقلمة تطبيقات الويندوز

تعنى الأقلمة عملية تعديل التطبيق بما يتناسب مع إحدى الثقافات. وتتكون عملية الأقلمة فى الأساس من ترجمة واجهة الاستخدام. ويوفر Visual Studio دعما كبيرا لعملية أقلمة تطبيقات نماذج الويندوز. أول متطلبات الأقلمة هى وجود ملفات موارد تخص الثقافة المحلية التى تتمثل فى اللغة، الأرقام، التاريخ، طريقة الكتابة، وغيرها من عناصر تكوين الثقافة. هناك طريقتان لتكوين هذه الملفات باستخدام بيئة التطوير المتكاملة (IDE) التى

تتوفر فى نظام NET Framework: الطريقة الأولى تتمثل فى قيام النظام بتوليد الملفات الخاصة بتحويل واجهة الاستخدام إلى الثقافة المحلية، ثم وضعها فى ملفات ضمن وحدة تجميع (Assembly) خاصة بها فى مشروع التطبيق. والطريقة الثانية هى إضافة قالب ملف موارد إلى المشروع ثم تدقيق هذا القالب باستخدام مصمم كود XML. والسبب الذى يدعو إلى استخدام الطريقة الثانية هو تحويل السلاسل التى تظهر فى مربعات الحوار ومربعات الرسائل إلى الثقافة المحلية. وفى هذه الحالة، يجب استخدام الكود للوصول إلى هذه الموارد.

التوليد التلقائي لملفات الموارد بواسطة Visual Studio

لتمكين Visual Studio من القيام بالتوليد التلقائي لملفات الموارد، ننفذ الخطوات التالية:

١. نكون تطبيق ويندوز جديد باسم "WindowsApplication1".

٢. فى نافذة Properties، نضبط خاصية Localizable على القيمة True.

٣. نسحب Button من جدول ToolBox إلى النموذج، ونضبط خاصية Text على "Hello World".

٤. نضبط خاصية Language بالنموذج على "Arabic (Egypt)".

٥. نضبط خاصية Text بمتحكم Button على "مرحبا أيها العالم".

٦. نحفظ المشروع ثم نقوم ببنائه.

٧. ننقر زر "Show All Files" فى مربع Solution Explorer. يؤدى ذلك إلى ظهور ملفات الموارد أسفل ملف Form1.vb أو ملف Form1.cs. ضمن الملفات، هناك ملف Form1.resx الذى يمثل الموارد الخاصة بالثقافة الأصلية المستخدمة فى بناء التطبيق. وهذا الملف يتم بناؤه داخل وحدة التجميع الأساسية بالمشروع. كما يوجد ضمن الملفات ملف Form1.ar-EG.resx يمثل الثقافة العربية فى مصر.

٨. نضغط مفتاح F5 لتشغيل التطبيق. سوف يترتب على ذلك ظهور مربع يحتوى على نص زر الأوامر باللغة الإنجليزية أو العربية بناءا على لغة واجهة الاستخدام فى نظام التشغيل.

ولكي يعرض التطبيق واجهة الاستخدام باللغة الخاصة بالثقافة المحلية، يجب أولا

ضبط ثقافة واجهة الاستخدام، كما يتضح من الكود التالى:

١. فى نافذة محرر الكود، نضيف الكود التالى فى بداية الوحدة وقبل الإعلان عن نموذجForm1:

```
Imports System.Globalization
Imports System.Threading
```

٢. نضيف الكود التالى فى إجراء New قبل استدعاء إجراء InitializeComponent:

```
Thread.CurrentThread.CurrentUICulture = New CultureInfo ("ar-EG")
```

٣. نحفظ التطبيق ثم نقوم ببنائه.

٤. نضغط مفتاح F5 لتشغيل التطبيق. يترتب على ذلك عرض نص زر التحكم باستخدام اللغة العربية.

إضافة وتدقيق ملفات الموارد يدويا إلى المشروع

لإضافة ملفات موارد يدويا إلى مشروع وتدقيق هذه الملفات، ننفذ الخطوات التالية:

١. فى قائمة Project، ننقر Add New Item.

٢. فى مربع حوار Add New Item، نختار قالب Assembly Resource File. نجعل اسم الملف يساوى "WinFormStrings.Resx" فى مربع Name. يتم إضافة هذا الملف تلقائيا فى مربع Solution Explorer وفتحة فى مصمم XML تحت ملصق Data.

٣. فى المصمم XML تحت ملصق Data، نختار Data فى جانب Data Tables.

٤. فى جانب Data، ننقر صفا خاليا وندخل "strMessage" فى عمود Name، وندخل عبارة "Hello World" فى عمود Value. ولا نحتاج إلى إدخال بيانات تحت عمود Type أو عمود mimeType بالنسبة للسلاسل لأن هذه الأعمدة تستخدم مع الكائنات. حيث يحتفظ عمود type بنوع بيانات الكائن، ويحتفظ عمود mimeType بأساس المعلومات الثنائية عندما يحتوى الكائن على معلومات ثنائية.

٥. فى قائمة File، ننقر Save WinFormStrings.resx لحفظة.

٦. نكرر الخطوات من ١ إلى ٥ لتكوين ملف يسمى "WinFormStrings.ar-EG.resx"، مع تغيير عبارة "Hello World" إلى عبارة "مرحبا أيها العالم".

للوصول إلى ملفات الموارد التى تم إضافتها بالتطبيق، ننفذ الخطوات التالية:

١. فى مربع تحرير الكود، نستورد مكتبة الأسماء System.Resources فى بداية وحدة الكود باستخدام الكود التالى:

```
Imports System.Resources
```

٢. فى مصمم النماذج، ننقر نقرا مزدوجا على متحكم Button لتكوين إجراء معالجة خاص بالنقر على هذا الزر ثم نضيف الكود التالى إلية:

نعلن عن مثل من تصنيف ResourceManager

```
Dim LocRM As New ResourceManager ("WindowsApplication1.WinFormStrings",
GetType (Form1). Assembly)
```

نخصص نص مفتاح "strMessage" لمربع رسالة

```
MessageBox.Show (LocRM.GetString ("strMessage"))
```

٣. نقوم ببناء وتشغيل التطبيق ثم ننقر على زر الأوامر بنموذج الويندوز. يترتب على ذلك، قيام التطبيق بعرض رسالة تحتوى على سلسلة نص مناسبة لثقافة واجهة الاستخدام. وعندما لا يجد التطبيق موردا خاصا بثقافة واجهة الاستخدام، يقوم بعرض النص المقابل فى الموارد الأساسية. ويجب ملاحظة أن التطبيق لن يعرض اللغة المطلوبة إلا بعد تحويل واجهة الاستخدام إلى تلك اللغة باستخدام الكود، كما سبق بيانه.

التنظيم الهرمي لاستخدام الموارد فى عملية الأقلمة

يتم تخزين الموارد الخاصة بعمليات استخدام الثقافة المحلية فى التطبيقات فى ملفات منفصلة. ويتم تحميل هذه الملفات على أساس ثقافة واجهة الاستخدام. ولفهم كيفية تحميل هذه الموارد، يمكن النظر إليها على أنها منظمة بطريقة هرمية. يوجد على قمة الهرم الموارد الأساسية أو الاحتياطية الخاصة بثقافة بناء التطبيق، مثل الثقافة الإنجليزية ("en"). ويتميز هذا النوع من الموارد بعدم وجود ملف خاص بها، حيث يتم تخزينها مباشرة فى وحدة التجميع الأساسية (Main Assembly). تحت الموارد الأساسية فى هذا النموذج الهرمي، توجد الموارد الخاصة بالثقافات الأخرى المحايدة. ويمكن تعريف الثقافة المحايدة بأنها الثقافة المرتبطة بلغة معينة وليس بإقليم. على سبيل المثال، اللغة العربية المحايدة يشار إليها بالرمز ("ar"). تحت موارد الثقافة المحايدة، تأتى موارد الثقافة المرتبطة بأحد الأقاليم، مثل اللغة العربية فى مصر التى يشار إليها بالرمز ("ar-EG").

وإذا حاول أحد التطبيقات تحميل أحد موارد الثقافة المحلية، مثل سلسلة نص، ولم يستطع العثور علية، سوف يقوم بالتجول داخل الهرم المذكور إلى أن يتم العثور عليها. وأحسن الطرق لتخزين الموارد، هى جعلها عامة إلى أقصى حد ممكن. يعنى ذلك تخزين سلاسل النصوص، الرسومات، وغيرها من موارد الثقافة المحلية فى ملفات موارد ثقافة محايدة بدلا من تخزينها فى ملفات موارد ثقافة إقليمية. على سبيل المثال، نفترض وجود موارد خاصة بالثقافة العربية فى إقليم مصر، وتوجد فوقها مباشرة فى هرم الموارد الثقافة الاحتياطية الخاصة بالتطبيق المتمثلة فى الثقافة الإنجليزية. إذا قام أحد المستخدمين بمحاولة استخدام اللغة العربية فى المملكة العربية السعودية التى يشار إليها بالرمز ("ar-SA")، فإن التطبيق سوف يستخدم الثقافة الاحتياطية بسبب عدم وجود ملف موارد خاص بموارد اللغة العربية فى المملكة العربية السعودية. وعلى العكس من ذلك، عندما يكون هناك ملف موارد محايد خاص باللغة العربية، فإن التطبيق سوف يستخدمه بغض النظر عن اللغة الإقليمية.

لغة الإدخال فى تطبيقات الويندوز

يمكن تعريف لغة الإدخال بأنها مخطط يتكون من الثقافة المحلية ولوحة المفاتيح، يتحكم فى تناظر المفاتيح الطبيعية الموجودة على لوحة المفاتيح مع حروف إحدى اللغات. ويحتوى نظام NET Framework على مجموعة من التصنيفات التى تحتوى على وسائل وخصائص للتعامل مع لغة الإدخال، أهمها ما يلى:

- تصنيف InputLanguage.
- تصنيف InputLanguageChangedEventArgs.
- تصنيف InputLanguageChangingEventArgs.

تصنيف InputLanguge

يحتوى هذا التصنيف على خصائص، وسائل، وحقول لإدارة لغة الإدخال. وفيما يلى عرض لأهم هذه الخصائص والوسائل واستخداماتها:

خاصية InstalledInputLanguages

تستخدم هذه الخاصية للحصول على قائمة بجميع لغات الإدخال المثبتة بالنظام. المثال التالى يستخدم مربع نص باسم textBox1 بعد ضبط خاصية MultiLine على القيمة True.

ويقوم بالحصول على قائمة بجميع لغات الإدخال المثبتة فى النظام:

```
Public Sub GetLanguages ()
   Dim lang As InputLanguage
   For Each lang In  InputLanguage.InstalledInputLanguages
      textBox1.Text &= lang.Culture.EnglishName & ControlChars.Cr
   Next lang
End Sub 'GetLanguages
```

خاصية Culture

تستخدم هذه الخاصية فى الحصول على الثقافة الخاصة بلغة الإدخال الحالية. وقيمة هذه الخاصية من نوع تصنيف CultureInfo الذى يحتوى على معلومات الثقافة الخاصة بلغة الإدخال الحالية، مثل اسم الثقافة، نظام الكتابة، التقويم المستخدم، ومعلومات أخرى. لتوضيح استخدام هذه الخاصية، يقوم المثال التالى بالحصول على لغة الإدخال الحالية ثم يستخدمها فى الحصول على معلومات الثقافة المرتبطة بها وعرضها فى مربع نص:

```
Public Sub MyCulture ()
   Dim myCurrentLanguage As InputLanguage =
   InputLanguage.CurrentInputLanguage
   Dim myCultureInfo As CultureInfo = myCurrentLanguage.Culture
   textBox1.Text = myCultureInfo.EnglishName
End Sub
```

خاصية CurrentInputLanguage

تستخدم هذه الخاصية فى الحصول على وضبط لغة الإدخال الخاصة بسلسلة العمليات الحالية. وقيمة هذه الخاصية من نوع تصنيف InputLanguage. المثال التالى يوضح كيفية الحصول على اسم لغة الإدخال الحالية، ويفترض وجود مربع نص باسم textBox1:

```
Public Sub MyCurrentInputLanguage ()
  Dim myCurrentLanguage As InputLanguage =
  InputLanguage.CurrentInputLanguage
  textBox1.Text = "Current input language is: " & _
     myCurrentLanguage.Culture.EnglishName
End Sub
```

خاصية Handle

تستخدم هذه الخاصية فى الحصول على مقبض لغة الإدخال. وقيمة هذه الخاصية من نوع مؤشر إلى رقم صحيح (IntPtr). المثال التالى يوضح كيفية الحصول على لغة الإدخال ثم

يستخرج قيمة خاصية Handle وطباعة النتيجة فى مربع نص:

```
Public Sub MyHandle ()
    Dim myCurrentLanguage As InputLanguage =
    InputLanguage.CurrentInputLanguage
       Dim myHandle As IntPtr = myCurrentLanguage.Handle
  textBox1.Text = "The handle number is: " & myHandle.ToString()
End Sub
```

خاصية LayoutName

تمكننا هذه الخاصية من الحصول على اسم مخطط لوحة المفاتيح الحالى، كما يظهر فى قيم الضبط الإقليمية بنظام تشغيل الويندوز. وقيمة هذه الخاصية من نوع String. المثال التالى، يوضح استخدام هذه الخاصية:

```
Public Sub MyLayoutName ()
    Dim myCurrentLanguage As InputLanguage =
    InputLanguage.CurrentInputLanguage
    If Not (myCurrentLanguage Is Nothing) Then
         textBox1.Text = "Layout: " & myCurrentLanguage.LayoutName
    Else
    textBox1.Text = "There is no current language"
   End If
End Sub
```

وسيلة Equals

تختبر هذه الوسيلة وجود المساواة بين لغتين من لغات الإدخال. وتتطلب تمرير كائن يمثل اللغة التى يتم اختبارها. ويترتب على تنفيذها إعادة قيمة منطقية (Boolean Value) تساوى True عند تحقق المقارنة، ويساوى False عند عدم تحققها.

وسيلة FromCulture

تعيد هذه الوسيلة لغة الإدخال المرتبطة مع ثقافة معينة، وتتطلب تمرير كائن من نوع تصنيف CultureInfo لتحديد الثقافة المستخدمة. ويترتب على نجاح تنفيذ هذه الوسيلة، الحصول على كائن يمثل لغة الإدخال المستعملة مع هذه الثقافة.

تصنيف InputLanguageChangedEventArgs

يوفر هذا التصنيف المعاملات اللازمة لحدث InputLanguageChanged. ويمكن استخدام البيانات التى نحصل عليها من هذا التصنيف فى اتخاذ قرارات خاصة بتغيير اتجاه النصوص لكى تصبح من اليمين إلى اليسار أو العكس. ويمكن أيضا استخدام هذه

المعلومات لتغيير خاصية CurrentCulture وخاصية CurrentUICulture المستخدمة فى تحويل التطبيقات نحو استخدام الثقافات المحلية. ويقوم هذا التصنيف بالتعرف على الثقافة المحلية ومجموعة الحروف المستخدمة فى لغة الإدخال الجديدة. ويحتوى هذا التصنيف على ثلاثة خصائص مهمة: خاصية CharSet، خاصية Culture، وخاصية InputLanguage.

خاصية CharSet

تستخدم هذه الخاصية فى الحصول على مجموعة الحروف المتعلقة بلغة الإدخال الجديدة. وتحتوى على قيمة من نوع Integer تمثل الرقم المميز لمجموعة الحروف، كما يتضح من الجدول رقم (١٨):

الرقم	مجموعة الحروف
0	ANSI_CHARSET
1	DEFAULT_CHARSET
2	SYMBOL_CHARSET
77	MAC_CHARSET
128	SHIFTJI_CHARSET
129	HANGEUL_CHARSET
129	HANGUL_CHARSET
130	JOHAB_CHARSET
134	GB2312_CHARSET
136	CHINESEBIG5_CHARSET
161	GREEK_CHARSET
162	TURKISH_CHARSET
163	VIETNAMESE_CHARSET
177	HEBREW_CHARSET
178	ARABIC_CHARSET
186	BALTIC_CHARSET
204	RUSSIAN_CHARSET
222	THAI_CHARSET
238	EASTEUROPE_CHARSET
255	OEM_CHARSET

جدول ١٨

خاصية Culture

تستخدم هذه الخاصية فى الحصول على الثقافة الخاصة بلغة الإدخال الجديدة، وتحتوى على قيمة من نوع تصنيف CultureInfo.

خاصية InputLanguage

تستخدم هذه الخاصية فى الحصول على قيمة تشير إلى لغة الإدخال الجديدة. وتحتوى على قيمة من نوع تصنيف InputLanguage.

حدث InputLanguageChanged

يقع هذا الحدث بعد تغيير لغة الإدخال. ويمكن استخدامه لإحداث تغييرات فى مظهر النموذج وفى النصوص التى تعتمد على لغة الإدخال. وتستقبل هذه الخاصية معاملا من تصنيف InputLanguageChangedEventArgs، الذى يحتوى على بيانات تتعلق بهذا الحدث. ويتم توفير هذه البيانات عن طريق خصائص تشمل: خاصية Culture، خاصية InputLanguage، وخاصية CharSet السابق إيضاحها.

يقوم المثال التالى بتكوين مربع نصوص زكيه (Rich Text Box) باستخدام الكود، ثم يقوم باستخدام تصنيف InputLanguageChangedEventArgs بصفته أحد المعاملات التى يتم تمريرها إلى إجراء معالجة حدث LanguageChange. يقوم هذا الإجراء بمعالجة حدث InputLanguageChanged للحصول على اسم لغة الإدخال التى تم التغيير إليها. ثم يقوم الإجراء بعد ذلك بمقارنة رمز لغة الإدخال مع رمز اللغة العربية. وعند تحقق الشرط، يتم تحويل اتجاه النصوص فى مربع النصوص الذكية ليبدأ من اليمين إلى اليسار.

```
Public Class Form1
   Inherits System.Windows.Forms.Form
   Dim WithEvents rtb As New RichTextBox ()
#Region " Windows Form Designer generated code "

   Public Sub New ()
      MyBase.New ()

      Me.Controls.Add (rtb)

      rtb.Dock = DockStyle.Fill
      InitializeComponent ()
   End Sub
```

```
Protected Overloads Overrides Sub Dispose (ByVal disposing As Boolean)
    If disposing Then
        If Not (components Is Nothing) Then
            components.Dispose ()
        End If
    End If

    MyBase.Dispose (disposing)

End Sub

Private components As System.ComponentModel.IContainer
<System.Diagnostics.DebuggerStepThrough ()> Private Sub
InitializeComponent()
    Me.AutoScaleBaseSize = New System.Drawing.Size (5, 13)
    Me.ClientSize = New System.Drawing.Size (292, 266)
    Me.Name = "Form1"
    Me.Text = "Form1"

End Sub

#End Region
Private Sub languageChanged ( _
        ByVal sender As Object, _
        ByVal e As InputLanguageChangedEventArgs _
    ) Handles MyBase.InputLanguageChanged

    If e.InputLanguage.Culture.TwoLetterISOLanguageName.Equals("ar") = True
    Then
        rtb.RightToLeft = RightToLeft.Yes
    End If
End Sub

End Class
```

تصنيف InputLanguageChangingEventArgs

يمكن استخدام المعلومات التى يوفرها هذا التصنيف فى تغيير اتجاه الكتابة بلغة الإدخال لكى تبدأ من اليمين إلى اليسار أو العكس. كما يمكن استخدام هذه المعلومات فى تغيير خاصية CurrentUICulture وCurrentCulture لكى يتم الحصول على موارد مختلفة. ولمنع لغة الإدخال من التغيير، نجعل قيمة خاصية Cancel فى هذا التصنيف تساوى True . وأهم الخصائص الموجودة فى هذا التصنيف هى: Cancel، Culture، InputLanguage،

وSysCharSet.

خاصية Culture

نستخدم Culture للحصول على معلومات عن الثقافة المحلية المرتبطة مع لغة الإدخال المطلوبة. وتحتوى هذه الخاصية على قيمة من نوع تصنيف CultureInfo.

خاصية InputLanguage

تستخدم خاصية InputLanguage فى الحصول على قيمة تشير إلى لغة الإدخال المطلوبة. وتحتوى هذه الخاصية على قيمة من نوع تصنيف InputLanguage.

خاصية مجموعة SysCharSet

تستخدم خاصية SysCharSet فى الحصول على قيمة توضح ما إذا كان طاقم الحروف الإفتراضى فى النظام يدعم طاقم الحروف الخاص بلغة الإدخال المطلوبة. وتحتوى هذه الخاصية على قيمة منطقية (Boolean Value).

حدث InputLanguageChanging

يقع هذا الحدث عندما يحاول المستخدم تغيير لغة الإدخال الخاصة بالنموذج. ويتطلب هذا الحدث تمرير معاملا من نوع تصنيف InputLanguageChangingEventArgs. يحتوى هذا المعامل على بيانات تتعلق بحدث محاولة تغيير لغة الإدخال الحالية، مثل خاصية Cancel، خاصية Culture، خاصية InputLanguage، وخاصية SysCharSet. ويماثل هذا الحدث حدث InputLanguageChanged، فيما عدا أن هذا الحدث يقع عند محاولة تغيير لغة الإدخال الحالية. ويمكن استخدام نفس المثال الموضح مع حدث InputLanguageChanged:

```
Public Class Form1
   Inherits System.Windows.Forms.Form
   Dim WithEvents rtb As New RichTextBox ()
#Region " Windows Form Designer generated code "

   Public Sub New ()
      MyBase.New ()

      Me.Controls.Add (rtb)
      rtb.Dock = DockStyle.Fill
```

```
      InitializeComponent ()
   End Sub
   Protected Overloads Overrides Sub Dispose (ByVal disposing As Boolean)
      If disposing Then
         If Not (components Is Nothing) Then
            components.Dispose ()
         End If
      End If

      MyBase.Dispose (disposing)

   End Sub

   Private components As System.ComponentModel.Icontainer
   <System.Diagnostics.DebuggerStepThrough ()> Private Sub
InitializeComponent()
      Me.AutoScaleBaseSize = New System.Drawing.Size (5, 13)
      Me.ClientSize = New System.Drawing.Size (292, 266)
      Me.Name = "Form1"
      Me.Text = "Form1"

   End Sub

#End Region
   Private Sub languageChangeing ( _
        ByVal sender As Object, _
        ByVal e As InputLanguageChangingEventArgs _
      ) Handles MyBase.InputLanguageChanging

      If    e.InputLanguage.Culture.TwoLetterISOLanguageName.Equals   ("ar")   =
      True Then
         rtb.RightToLeft = RightToLeft.Yes
      End If
   End Sub

End Class
```

تطبيق على تغيير لغة الإدخال بتطبيقات الويندوز

المثال التالى يعرض برنامجا كاملا عن استخدام لغة الإدخال. يقوم البرنامج بالوظائف التالية:

- الحصول على جميع لغات الإدخال المثبتة بالنظام ووضع رموزها فى مربع سرد (List Box).

- استخدام معالج لحدث النقر المزدوج على بنود مربع السرد للحصول على رمز اللغة الذى يتم اختياره.
- استخدام الرمز المستخرج من مربع السرد للحصول على اللغة المقابلة من بين لغات الإدخال المثبتة.
- جعل اللغة التى تم اختيارها لغة الإدخال الحالية (CurrentInputLanguage).
- استخدام حدث InputLanguageChanged لتحويل اتجاه الكتابة من اليمين إلى اليسار أو العكس.

```
Public Class Form1
   Inherits System.Windows.Forms.Form

[" Windows Form Designer generated code "]
   Private Sub languageChange( _
        ByVal sender As Object, _
        ByVal e As InputLanguageChangedEventArgs _
     ) Handles MyBase.InputLanguageChanged
```

١. تغيير اتجاة الكتابة إذا كانت اللغة المستخدمة هى اللغة العربية:

```
 If  e.InputLanguage.Culture.TwoLetterISOLanguageName.Equals  ("ar")  =  True
Then
         Me.RightToLeft = RightToLeft.Yes
      Else
         Me.RightToLeft = RightToLeft.No
      End If
   End Sub

   Private  Sub  Form1_Load  (ByVal  sender  As  System.Object,  ByVal  e  As
System.EventArgs) Handles MyBase.Load
```

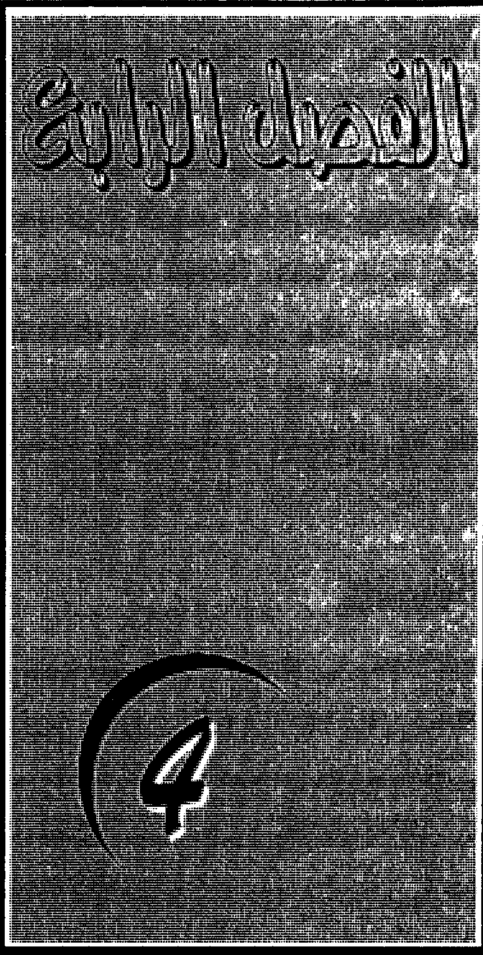
٢. تأهيل مربع سرد بجميع لغات الإدخال المثبتة بالكمبيوتر.

```
      Dim lang As InputLanguage
      For Each lang In lang.InstalledInputLanguages
         ListBox1.Items.Add (lang.Culture.TwoLetterISOLanguageName)
      Next
   End Sub
```

٣. تغيير لغة الإدخال إلى اللغة التى يختارها المستخدم من مربع السرد.

```
   Private  Sub  ListBox1_DoubleClick  (ByVal  sender  As  Object,  ByVal  e  As
```

```
System.EventArgs) Handles ListBox1.DoubleClick
        Dim ToLtr As String
        Dim lang As InputLanguage
        ToLtr = ListBox1.SelectedItem
        For Each lang In lang.InstalledInputLanguages
            If lang.Culture.TwoLetterISOLanguageName = ToLtr Then
                InputLanguage.CurrentInputLanguage = lang
            End If
        Next
    End Sub
End Class
```



تكوين واجهات استخدام التطبيقات

واجهة التعامل أو الواجهة البينية (Interface) هى البوابة الرئيسية التى تمكننا من استخدام التطبيق عن طريق طلب تنفيذ المهام المختلفة التى يمكن أن يقوم بها التطبيق، إدخال المعلومات التى يتطلبها تنفيذ هذه المهام، وقراءة وطباعة البيانات المختلفة الناتجة عن التنفيذ. وتتكون واجهات تعامل تطبيقات الويندوز من عناصر رسومية مرئية وغير مرئية، تشتمل أساسا على النموذج (Form) وعلى أدوات التحكم المختلفة. يمثل النموذج لوحة منبسطة ذات خصائص ووظائف متنوعة، توضع عليها أدوات تحكم (Controls) متنوعة. بينما تمثل أدوات التحكم كائنات رسومية تقوم بمهمة إدارة روابط التطبيق، استقبال طلبات المستخدمين، استقبال المعلومات التى يدخلونها، وعرض نتائج التشغيل المختلفة.

استخدام أدوات تحكم نماذج الويندوز

أثناء تصميم وتعديل واجهات التعامل (Interfaces) فى الحلول التى نقوم بتكوينها، سوف يكون من الضروري استخدام أدوات التحكم (Controls) المختلفة. ويمكن تعريف أدوات التحكم بأنها كائنات توضع على النماذج للقيام بوظائف محددة. ويمتلك كل نوع من أدوات التحكم مجموعة من الخصائص، الوسائل، والأحداث التى تجعله مناسبا للاستخدام فى غرض معين. ويمكن التعامل مع هذه الأدوات فى وقت التصميم، كما يمكن كتابة الكود اللازم للتعامل معها أتوماتيكيا فى وقت التشغيل. والمكان الذى توجد به أدوات التحكم فى Visual Studio، هو مربع ToolBox. ويمكن عرض ToolBox بالنقر على قائمة View ثم اختيار ToolBox. يعرض مربع ToolBox مجموعة متنوعة من الأدوات التى يمكن استخدامها فى مشروعات Visual Basic المختلفة.

يحتوى مربع ToolBox على عدد من الملصقات (Tabs) المرتبطة بصفحات تحتوى على أدوات تحكم مختلفة. وتتغير الملصقات وما تحتويه من أدوات بناءا على نوع المصمم والمحرر المستخدم فى Visual Studio. ويمكن أن تشمل البنود المعروضة كائنات NET، كائنات COM، كائنات HTML، أجزاء من الكود، ونصوص. ويعرض مربع ToolBox ملصقين بصفة دائمة: ملصق General، وملصق Clipboard Ring. ثم يتم بعد ذلك عرض

ملصقات وأدوات أخرى بناءا على نوع المصمم أو المحرر المستخدم. ويمكن أيضا إضافة ملصقات خاصة بنا إلى مربع ToolBox. ويمكن تلخيص الملصقات التى يحتوى عليها مربع ToolBox فيما يلى:

- الملصق العام (General Tab). يظهر هذا الملصق دائما عند فتح مربع ToolBox. ويمكن استخدامه لتخزين أدوات التحكم الافتراضية المستخدمة فى المشروعات المختلفة، بما فى ذلك أدوات التحكم التى يكونها المستخدم. وفى البداية لا يحتوى هذا الملصق إلا على المؤشر (Pointer). ويمكن إضافة، حذف، إعادة تسمية، وترتيب البنود فى هذا الملصق. ويمكن سحب هذه البنود من الملصق إلى وجه المصمم أو المحرر المستخدم.
- ملصق البيانات (Data Tab). يعرض ملصق Data Tab كائنات البيانات التى يمكن إضافتها إلى النماذج والمركبات. ويظهر هذا الملصق عند تكوين مشروع مرتبط بمصمم.
- ملصق المكونات (Components Tab). يعرض مكونات يمكن إضافتها إلى أدوات التصميم فى Visual Basic. وبالإضافة إلى المكونات التى يحتوى عليها نظام NET Framework، يمكن إضافة المكونات التى يعدها المستخدم أو طرف أخر.
- ملصق XML (XML Tab). يعرض هذا الملصق العناصر التى يمكن إضافتها إلى مخططات XML وفئات البيانات فى ADO.NET عندما يكون مصمم XML معروضا فى مشهد المخطط (Schema View). ويصبح هذا الملصق متاحا عند العمل مع ملف امتدادة xsd.
- ملصق لوحة القص (Clipboard Ring Tab). فى هذا الملصق، يتم تخزين العناصر الإثنى عشرة الأخيرة التى تم إضافتها إلى لوحة القص (Clipboard) باستخدام أمر Cut أو أمر Copy أثناء العمل مع مصمم أو محرر كود. يمكن بعد ذلك سحب هذه البنود من هذا الملصق ووضعها على المصمم أو المحرر الجاري استخدامه.
- ملصق نماذج الويندوز (Windows Forms Tab). يعرض هذا الملصق قائمة بأدوات تحكم

نماذج الويندوز ومربعات الحوار التى يمكن استخدامها فى تطبيقات الويندوز. ويتم ترتيب أدوات التحكم والمكونات الموجودة على هذا الملصق بناءا على تكرار الاستخدام. ولتغيير الترتيب على أساس حروف الهجاء، ننقر بزر الماوس الأيمن على مربع ToolBox ثم نختار Sort Items Alphabetically. وبنود هذا الملصق هى البنود التى سوف نوضحها فى هذا الفصل.

تحديد وضع أدوات التحكم على النماذج

بعد تكوين النموذج، تكون الخطوة التالية فى تكوين واجهات التعامل فى تطبيقات الويندوز هى تحديد وضع أدوات التحكم على ذلك النموذج. تشمل هذه المهمة الوظائف التالية: إضافة أدوات التحكم إلى النموذج، ترتيب أدوات التحكم على النموذج، وتمييز أدوات التحكم المختلفة. وتشتمل كل وظيفة من هذه الوظائف على كثير من التفصيلات التى يجب إيضاحها فى الموضوعات التالية.

إضافة أدوات التحكم إلى نماذج الويندوز

هناك نوعان من أدوات التحكم التى يمكن إضافتها إلى نموذج الويندوز لبناء الواجهة البينية بين التطبيق وبين المستخدم: أدوات تحكم تحتوى على واجهة بينية (Controls)، وأدوات تحكم لا تحتوى على واجهة بينية (Components). ويختلف الموقع المستخدم لإضافة كل نوع من هذين النوعين على النموذج.

إضافة أدوات التحكم ذات الواجهة البينية

لرسم أداة تحكم، ننفذ الخطوات التالية:

١. نفتح النموذج.

٢. ننقر أداة التحكم التى نريد إضافتها للنموذج فى مربع ToolBox.

٣. على سطح النموذج، ننقر الموقع الذى نريد وضع الركن اليسار الأعلى للمتحكم به، ثم نسحب الماوس إلى المكان الذى نريد أن يكون به الركن الأسفل الأيمن للمتحكم. يترتب على ذلك إضافة المتحكم إلى النموذج فى الموقع الذى تم تحديد.

ولكل متحكم حجم افتراضي محدد يمكن استخدامه عن طريق سحب أداة التحكم من مربع

الأدوات إلى سطح النموذج:

١. نفتح النموذج فى مصمم النماذج.

٢. فى مربع ToolBox، ننقر على المتحكم الذى نريد إضافته و نسحبة إلى النموذج. يترتب على ذلك إضافة المتحكم إلى النموذج فى الموقع المحدد باستخدام حجمة الإفتراضى. ويمكن النقر المزدوج على المتحكم لإضافة المتحكم بحجمه الإفتراضى فى أقصى أعلى يسار النموذج.

ويمكـن إضـافة أداة تحكـم ديناميكيا أثناء التشغيل باستخدام الكود. فى المثال التالى، سوف يتم إضافة مربع نص إلى نموذج عند النقر على أحد الأزرار باستخدام إجراء معالجة النقر على أحد أزرار الأوامر:

```
Private Sub Button1_Click (ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
  Dim MyText As New TextBox ()
  MyText.Location = New Point (25, 25)
  Me.Controls.Add (MyText)
End Sub
```

إضافة مكون بدون واجهه بينية

على خلاف أدوات التحكم الأخرى، لا تقدم المكونات واجهات للتعامل مع المستخدم. وعندما يجرى إضافة مكون إلى نموذج، يقوم مصمم نماذج الويندوز بعرض مربع متغير الحجم أسفل النموذج لوضع المكونات علية يطلق علية Components Tray. وبمجرد إضافة مكون إلى هذا المربع، يمكننا اختياره وضبط خصائصه مثلما نفعل مع أدوات التحكم الموجودة على النموذج. ولإضافة متحكم إلى نموذج، نتبع الخطوات التالية:

١. نفتح النموذج المطلوب فى مصمم نماذج الويندوز.

٢. فـى مـربع ToolBox، نـنقر عـلى المكـون و نسـحبة إلى النموذج. يترتب على ذلك ظهور المكون فى مربع المكونات أسفل مصمم النماذج.

ويمكن إضافة المكونات إلى النماذج فى وقت التشغيل. ويعتبر ذلك من الإجراءات الشائعة فى ضوء حقيقة أن المكونات ليس لها واجهات بينية مرئية. فى المثال التالى، نضيف مكون ColorDialog إلى النموذج وقت التشغيل:

```
Dim cdlgTest As New ColorDialog ()
cdlgTest.ShowDialog ()
```

ترتيب أدوات التحكم على نماذج الويندوز

لجعل مخطط واجهة التعامل فى تطبيق ويندوز يتطابق مع المعايير الموضوعة، يجب ترتيب أدوات التحكم الموضوعة على سطح النموذج. يشمل هذا الترتيب عمليات، مثل التصفيف (Aligning)، التثبيت (Anchoring)، النسخ (Copying)، إستقرار (Docking)، وضع الكائنات فى طبقات (Layering)، الإقفال (Locking)، تحديد الإحداثيات (Positioning)، تغيير الحجم (Resizing)، وجدولة أدوات التحكم (Tab Order).

تصفيف أدوات التحكم على النموذج

١. نفتح النموذج الذى يحتوى على أدوات التحكم التى نريد ترتيب أوضاعها فى مصمم النماذج.

٢. نختار أدوات التحكم التى نريد تصفيفها على أساس المتحكم الذى يتم اختياره أولا.

٣. فى قائمة Format، نشير إلى Align ثم ننقر واحدا من الخيارات السبعة الموجودة.

تثبيت أدوات التحكم على نموذج الويندوز

عند تصميم أحد النماذج التى تمكن المستخدم من التحكم فى حجمة وقت التشغيل، يجب أيضا أن تتغير أحجام وأماكن أدوات التحكم بالتناسب. لتغيير حجم أدوات التحكم ديناميكيا عند تغيير حجم النموذج، نستخدم خاصية Anchor فى هذه الأدوات. تحدد هذه الخاصية مكانا لتثبيت أداة التحكم على النموذج. وعندما يتم التثبيت، يحتفظ المتحكم بالمسافة بينة وبين حدود النموذج عند تغيير حجم النموذج. على سبيل المثال، نفترض وجود متحكم من نوع TextBox مثبت بالنسبة للحدود اليسرى واليمنى والسفلية للنموذج. عند تغيير حجم النموذج، تقوم أداة التحكم المذكورة بتغيير حجمها لكى تحتفظ بنفس المسافات مع حدود النموذج المستخدمة فى التثبيت. وعندما لا يتم تثبيت أداة التحكم، تتغير المسافات بينها وبين حدود النموذج عند تغيير حجم هذا النموذج. لتثبيت أحد أدوات التحكم على نموذج:

١. نختار أداة التحكم التى نريد تثبيتها. ويمكن تثبيت أدوات تحكم متعددة فى نفس الوقت بالضغط على مفتاح CTRL والنقر على كل متحكم لاختياره ثم اتباع باقى الخطوات التالية.

٢. فى نافذة Properties، ننقر السهم الذى على يمين خاصية Anchor. يترتب على ذلك، ظهور مربع به صليب. ننقر على القمة، اليسار، اليمين، أو القاع بالنسبة للصليب. يؤدى ذلك إلى تثبيت أدوات التحكم تلقائيا بالنسبة لما تم اختياره. ولحذف جانب من جوانب التثبيت، ننقر على ذراع الصليب فى ذلك الاتجاه.

٣. ننقر خاصية Anchor مرة أخرى لإقفال مربع التثبيت.

عند عرض النموذج وقت التشغيل، يتغير حجم أداة التحكم لكى تحتفظ بنفس المسافات مع حدود النموذج. وهناك بعض أدوات التحكم، مثل ComboBox، يكون ارتفاعها مقيدا. مثل هذه الأدوات لا يترتب على تثبيتها زيادة ارتفاعها عن الارتفاع الإفتراضى لها.

نسخ أدوات التحكم

يمكن نسخ متحكم إلى نفس النموذج، إلى نموذج أخر فى نفس المشروع، أو إلى لوحة القص بغرض استخدامه فى حلول أخرى. ولنسخ أداة تحكم، ننفذ الخطوات التالية:

١. نختار المتحكم، ثم نختار Copy من قائمة Edit.

٢. يمكن لصق المتحكم الذى تم اختياره فى أى نموذج يقبل هذا النوع من أدوات التحكم.

استقرار النموذج

يمكن جعل أداة تحكم تأخذ جانب أحد حدود النموذج أو تشغل النموذج بالكامل. والمقصود بالاستقرار (Docking)، جعل الأداة تستقر بجانب حافة محددة مثل رسو السفن بجانب الأرصفة. على سبيل المثال، يقوم نظام ويندوز اكسبلورر بإرساء متحكم TreeView على الجانب الأيسر من النافذة ومتحكم ListView على الجانب الأيمن منها. ويتم تحديد نمط الاستقرار لجميع أدوات التحكم ذات الواجهات المرئية باستخدام خاصية Dock. لوضع

أداة تحكم فى حالة استقرار، نتبع الخطوات التالية:

١. نختار أداة التحكم التى نريد إرسائها.

٢. فى نافذة Properties، ننقر السهم الذى على يمين خاصية Dock. يترتب على ذلك عرض نافذة تحرير تبين سلسلة من المربعات التى تمثل حواف ومركز النموذج.

٣. ننقر الزر الذى يمثل حافة النموذج التى نريد الاستقرار بجانبها. ولشغل كامل الحاوية، ننقر المربع الذى فى المركز. ولحجب عملية الاستقرار، ننقر None. ويترتب على عملية الاستقرار تغيير أحجام أدوات التحكم لتتلاءم مع حدود الحافة التى يتم عندها الاستقرار.

وضع أدوات التحكم فى طبقات بنماذج الويندوز

عند تكوين واجهة استخدام مركبة أو العمل مع نموذج متعدد الوثائق (MDI)، يكون من المرغوب فيه غالبا، وضع أدوات التحكم والنماذج التابعة فى طبقات (Layering) لتكوين واجهات استخدام أكثر تعقيدا. ولتحريك ومتابعة أدوات التحكم والنماذج داخل مجموعة، نتعامل مع Z-order الخاص بها. ويمكن تعريف Z-order بأنها عملية وضع أدوات التحكم فى طبقات مرئية على نموذج، على امتداد Z-order (محور العمق) الخاص بها. بحيث تأتى النافذة التى فى أعلى الترتيب فوق جميع النوافذ الأخرى، وتأتى جميع النوافذ الأخرى فوق النافذة التى فى قاع الترتيب. ولوضع أدوات التحكم فى طبقات وقت التصميم، نتبع الخطوات التالية:

١. نختار المتحكم المطلوب.

٢. من قائمة Format، نشير إلى Order ثم ننقر Bring To Front أو Send To Back.

ولوضع أدوات التحكم فى طبقات باستخدام الكود:

نستخدم وسيلة BringToFront ووسيلة SendToBack للتعامل مع الترتيب باتجاه العمق (Z-order). على سبيل المثال، إذا كان هناك متحكم TextBox باسم txtFirstName يقع تحت متحكم أخر ونريد وضعة على القمة، نستخدم الكود التالى:

```
txtFirstName.BringToFront ()
```

وبالنسبة لأدوات التحكم التى تحتويها أداة تحكم أخرى مثل مربع المجموعة (GroupBox)، يمكن أيضا وضع أدوات التحكم فى طبقات وتحريك أدوات التحكم والمجموعة معا.

لإقفال أدوات التحكم

يمكن إقفال أوضاع أدوات التحكم على نموذج بمجرد وضعها بطريقة صحيحة لكى نمنع تحريكها أو تغيير أحجامها بطريقة غير مقصودة. لإقفال أداة تحكم على نموذج ويندوز، نتبع ما يلى:

١. نختار أداة التحكم على نموذج الويندوز.

٢. فى نافذة الخصائص، ننقر على خاصية Locked ثم نختار True.

بالإضافة إلى ذلك، يمكننا إقفال أدوات التحكم على النموذج فى الحال. مما يقدم مساعدة كبيرة عندما يحتوى النموذج على العديد من أدوات التحكم. للقيام بذلك، ننفذ ما يلى:

- فى قائمة Format، نختار Lock Controls. مما يترتب علية إقفال جميع أدوات التحكم وإقفال حجم النموذج أيضا لأن النموذج متحكم أيضا.

تحديد إحداثيات أدوات التحكم على النموذج

لتحديد مواقع أدوات التحكم، يمكن تنفيذ ذلك يدويا، باستخدام مصمم نماذج الويندوز، أو باستخدام الكود لتحديد قيمة خاصية Location.

لتحديد موقع المتحكم يدويا على سطح النموذج:

- نسحب أداة تحكم إلى الموقع المناسب على سطح النموذج باستخدام الماوس. ولتحديد الموقع بدقة أكثر، يمكن اختيار المتحكم ثم تحريكه باستخدام مفاتيح الأسهم بلوحة المفاتيح.

ولتحديد موقع أداة تحكم باستخدام نافذة الخصائص:

١. نختار أداة التحكم على سطح النموذج.

٢. فى نافذة Properties، ندخل قيم الإحداثيات بخاصية Location مع فصلها باستخدام الفاصلة. القيمة الأولى (X) هى المسافة من الحد الأيسر للحاوية، والقيمة الثانية (Y) هى المسافة من الحد الأعلى للحاوية، ويتم القياس باستخدام وحدة بكسل.

ولوضع متحكم على نموذج باستخدام الكود، نضبط خاصية Location فى المتحكم على قيمة Point. الكود التالى يحدد موقع جديد لمتحكم Button على سطح النموذج:

```
Button1.Location = New Point (100, 100)
```

ولزيادة أبعاد موقع متحكم باستخدام الكود، نستخدم خاصية Left لزيادة قيمة الإحداثي X الخاص بموقع المتحكم، كما يتضح من الكود التالى:

```
Button1.Left += 200
```

تغيير حجم أدوات التحكم فى نماذج الويندوز

لتغيير حجم أدوات التحكم باستخدام مصمم نماذج الويندوز:

- ننقر على المتحكم فى مصمم النماذج ثم نسحب أحد مقابض تغيير الحجم الثمانية به. ويمكن تغيير الحجم عن طريق اختيار أداة التحكم ثم الضغط على مفتاح ARROW أثناء الضغط على مفتاح SHIFT.

ويمكن تغيير حجم أدوات التحكم باستخدام الكود:

١. ندخل الكود التالى لتكوين مستطيل ومتحكم Button فى إجراء خاص بذلك.

```
Private Sub MakeSizedButton ()
  Dim rect1 as New Rectangle (50, 100, 75, 23)
  Dim Button1 as New Button ()
End Sub
```

٢. نضيف كود لضبط خاصية Bounds فى متحكم Button1 ليتساوى مع أبعاد حجم المستطيل.

```
Button1.Bounds = rect1
```

ولتغيير حجم عدد من أدوات التحكم على نموذج ويندوز:

١. نختار أدوات التحكم بالنقر على كل منها أثناء النقر على مفتاح SHIFT.
٢. فـى قائمـة Format، نشـير إلى Make Same Size، ثـم نختار واحدا من الخيارات المتاحة.

جدولة أدوات التحكم على نماذج الويندوز

جدولة أدوات التحكم هى ترتيب التنقل بين هذه الأدوات عندما يقوم المستخدم بالضغط على مفتاح TAB. ويحتوى كل نموذج على ترتيب خاص به. وفى الوضع الإفتراضى، يكون ترتيب الجدولة هو ترتيب تكوين أدوات التحكم على نموذج الويندوز. ويبدأ الترتيب بالرقم صفر. ولضبط ترتيب الجدولة الخاصة بأداة تحكم، نتبع الخطوات التالية:

١. فى قائمة View، نختار Tab Order. يؤدى ذلك إلى ظهور ترتيب الجدولة على النموذج، الذى يتمثل فى ظهور رقم فى الركن الأعلى على اليسار بكل أداة تحكم تمثل قيمة خاصية TabIndex.
٢. ننقر على أدوات التحكم بالترتيب لبناء ترتيب الجدولة الجديد. ويمكن ضبط قيمة ترتيب المتحكم الموضوعة فى ترتيب جدولة لتكون صفر أو أكبر من الصفر. وعند حدوث تكرار، يتم تقييم ترتيب العمق وجعل ترتيب المتحكم الذى فى القمة يسبق ترتيب المتحكم التالى له فى العمق.
٣. عند الانتهاء من عملية الترتيب، نختار Tab Order مرة أخرى من قائمة View لإقفال هذا المحرر.

ويجب ملاحظة أن أدوات التحكم التى لايمكن أن تحوز بؤرة التركيز، مثل أدوات التحكم غير المرئية أو المحجوبة، لا يوجد بها خاصية TabIndex ولا يتم إدراجها فى ترتيب الجدولة. وعندما يقوم المستخدم بالضغط على مفتاح TAB، يتم تجاوز هذه الأدوات. ويمكن ضبط ترتيب الجدولة فى نافذة Properties باستخدام خاصية TabIndex التى تحدد موقع المتحكم فى ترتيب الجدولة.

ومع أن متحكم GroupBox لا يحوز التركيز، إلا أن له رقم جدولة يستخدم فى ترتيب

جدولة أدوات التحكم التى بداخلة. ويتم ترقيم جدولة الأدوات التى بداخل GroupBox على أساس رقم يتكون من رقم المجموعة بالإضافة إلى رقم الترتيب داخل المجموعة. على سبيل المثال، إذا كان رقم المجموعة هو الرقم 8 فإن رقم أول متحكم بداخلها سوف يكون 8.0 ورقم المتحكم الثانى 8.1 وهكذا .

ولحذف متحكم من ترتيب الجدولة، نضبط خاصية TabStop بالمتحكم ذات العلاقة على القيمة False باستخدام نافذة Properties. ويستمر المتحكم المحذوف من الترتيب فى الاحتفاظ بترتيبه فى الجدولة على الرغم من تجاوزة عند التنقل بين أدوات التحكم باستخدام مفتاح TAB. ويجب ملاحظة أنة عند استخدام مفتاح TAB، يكون هناك توقف واحد لمجموعة أزرار الراديو (Radio Group). ويتم ضبط خاصية TabStop لزر الراديو الذى يتم اختياره على القيمة True تلقائيا، وتبقى هذه الخاصية False لباقى أزرار الراديو.

انحياز أدوات التحكم إلى شبكة نموذج الويندوز

يمكن استخدام خاصية SnapToGrid للتحكم فى مواقع أدوات التحكم على شبكة نموذج الويندوز أو تحريكها علية. ويترتب على جعل قيمة هذه الخاصية True، انحياز أدوات التحكم إلى صفوف وأعمدة شبكة مصمم النماذج.

تمييز أدوات التحكم

تقوم أدوات تحكم نماذج الويندوز بعرض بعض النصوص ذات العلاقة بالوظيفة الأساسية للأداة. على سبيل المثال، يعرض متحكم Button دائما عنوانا يشير إلى نوع الفعل الذى يمكن تنفيذه عند النقر على الزر. ويمكن بالنسبة لجميع أدوات التحكم، ضبط هذا النص أو إعادة قيمته باستخدام خاصية Text، كما يمكن تغيير طاقم الحروف باستخدام خاصية Font. لضبط النص وطاقم الحروف المستخدم فى متحكم باستخدام مصمم النماذج:

١. نختار المتحكم فى نافذة تصميم النماذج.

٢. فى نافذة Properties، نضبط خاصية Text على النص المناسب. ولتكوين مفتاح وصول إلى المفتاح، نضع الرمز & قبل أحد حروف النص.

٣. نضبط خاصية Font في نافذة Properties على طاقم الحروف الذى نريد استخدامه.

ولضبط النص فى متحكم باستخدام الكود:

١. نضبط خاصية Text باستخدام الكود التالى:

```
Button1.Text = "Click here to save changes"
```

٢. نضبط خاصية Font باستخدام الكود التالى:

```
Button1.Font = New Font ("Arial", 10, FontStyle.Bold, GraphicsUnit.Point)
```

ضبط الصورة التى تعرضها أداة تحكم

هناك العديد من أدوات التحكم التى يمكنها عرض الصور. ويمكن أن تكون الصورة أيقونة تعبر عن الغرض من المتحكم، كما يتضح من عرض صورة الطابعة على زر الطباعة. ويمكن أن تكون الصورة خلفية تكون تأثيرا محددا للمتحكم على المستخدم. ويمكن ضبط الصورة فى جميع أدوات التحكم التى تعرض الصور، باستخدام خاصية Image أو خاصية BackGroundImage.

لضبط الصورة التى يعرضها أحد النماذج باستخدام مصمم النماذج:

١. نختار المتحكم فى مصمم النماذج.

٢. فى نافذة Properties، نختار خاصية Image أو BackgroundImage الخاصة بالمتحكم ثم ننقر على زر نقاط الحذف (Ellipsis Button) لعرض مربع حوار Open.

٣. نختار ملف الصورة الذى نريد عرضة.

ويمكن ضبط الصورة باستخدام الكود التالى:

```
Dim path As String = "C:\Windows\zapotec.bmp"
Button1.Image = Image.FromFile (path)
```

أو استخدام الكود التالى:

```
Button1.BackgroundImage = BackgroundImage.FromFile (path)
```

تكوين مفاتيح الوصول لأدوات التحكم بنماذج الويندوز

يمكن تعريف مفتاح الوصول بأنة حرف تحته خط فى نص قائمة (Menu)، بند قائمة (MenuItem)، أو نص تمييز لأحد أدوات التحكم. هذه المفاتيح تمكن المستخدم من النقر

على زر عن طريق ضغط مفتاح ALT بالترافق مع مفتاح وصول سابق التعريف. على سبيل المثال، نفترض وجود زر يقوم بتشغيل إجراء لطباعة نموذج ويحتوى على النص "Print". يترتب على وضع الرمز & قبل الحرف "P" وضع خط أسفل هذا الحرف عند تشغيل التطبيق. ويمكن الضغط على مفتاح الحرف P أثناء الضغط على مفتاح ALT لتنفيذ عملية الطباعة. ولا يمكن ضبط مفتاح وصول لمتحكم لا يستطيع أن يحوز بؤرة التركيز عند استخدام مفتاح TAB.

لتكوين مفتاح وصول إلى متحكم باستخدام مصمم النماذج:

١. نختار المتحكم.

٢. فى نافذة Properties، نضبط خاصية Text على سلسلة تشتمل على الرمز & قبل الحرف الذى نريد استخدامه للوصول إلى المتحكم.

ولتكوين مفتاح وصول خاص بمتحكم باستخدام الكود، نضبط خاصية Text على سلسلة تحتوى على الرمز & قبل الحرف المخصص للوصول إلى المفتاح، كما يتضح من الكود التالى:

```
Button1.Text = "&Print"
```

ولوضع رمز & فى نص بدون تكوين مفتاح وصول، ندخل رمزين متتابعين (&&). يترتب على ذلك اشتمال النص على أحد الرمزين مع عدم وضع خط أسفل الحرف الذى يأتى بعدة.

أنواع أدوات تحكم نماذج الويندوز

يمكن تقسيم أدوات تحكم نماذج الويندوز إلى قسمين رئيسيين: أدوات تحكم الويندوز التى بها واجهات تعامل مع المستخدمين ، وأدوات تحكم الويندوز التى لا تحتوى على واجهات تعامل مع المستخدمين. وتعرف أدوات تحكم الويندوز التى بها واجهات تعامل مع المستخدمين بالمتحكمات (Controls)، بينما تعرف أدوات تحكم الويندوز التى ليس لها واجهات تعامل مع المستخدمين بالمكونات (Components). بالإضافة إلى هذا التقسيم، يمكن تقسيم أدوات التحكم أيضا على أساس الوظائف التى تقوم بها.

أدوات التحكم ذات واجهات التعامل (Controls)

تظهر أدوات التحكم ذات واجهات التعامل مع المستخدمين (Controls) على سطح نموذج الويندوز مباشرة، وتعتبر جزءا أساسيا فى الواجهات البينية بين المستخدم وبين التطبيقات. وتقوم هذه الأدوات بعرض المعلومات أمام المستخدم، استقبال المعلومات التى يدخلها المستخدم إلى التطبيق، كما أنها تمثل الأدوات التى تتفاعل مباشرة مع الأحداث التى يسببها المستخدم. فيما يلى شرح للأدوات التى تكون هذا النوع.

الزر (Button)

يعرف هذا المتحكم أيضا باسم زر الأوامر (Command Button)، ويسمح للمستخدم بالنقر علية لتنفيذ أحد الأفعال. ويمكن لهذا المتحكم أن يعرض كلا من النصوص والصور. وعندما يقوم المستخدم بالنقر على زر، يتم استدعاء إجراء معالجة حدث النقر (Click). ونقوم بوضع كود فى إجراء معالجة الحدث لتنفيذ بعض المهام التى يتم اختيارها. وتحتوى خاصية Text بمتحكم الزر على النص الذى يجرى عرضة على الزر. وإذا كان النص يزيد على طول الزر، يلتف النص على السطر التالى بالزر. ويمكن أن تحتوى خاصية Text على مفتاح وصول (Access Key)، مما يسمح للمستخدم بالضغط على المتحكم عن طريق الضغط على مفتاح الوصول ومفتاح ALT معا. ويتم التحكم فى شكل النص باستخدام خاصية Font وخاصية TextAlign. ويمكن أن يعرض الزر رسم من الرسومات باستخدام خاصية Image أو خاصية ImageList.

تخصيص أحد الأزرار ليكون الزر الإفتراضى بالنموذج

يمكن تخصيص أحد أزرار النموذج ليكون الزر الإفتراضى. يؤدى ذلك إلى تنفيذ عملية النقر على الزر عندما يتم الضغط على مفتاح Enter بغض النظر عن المتحكم الذى يحوز بؤرة التركيز على النموذج. ويتم مخالفة هذه القاعدة عندما يكون هناك زر أخر يحوز بؤرة التركيز. فى هذه الحالة يؤدى الضغط على مفتاح Enter، إلى النقر على الزر محل التركيز. لتحديد الزر الإفتراضى باستخدام مصمم النماذج:

١. نختار النموذج فى مصمم النماذج.

٢. في نافذة Properties، نضبط خاصية AcceptButton على اسم الزر المستهدف.

ولتحديد الزر الإفتراضي باستخدام الكود، نستخدم الكود التالي:

```
Private Sub SetDefault (ByVal myDefaultBtn As Button)
 Me.AcceptButton = myDefaultBtn
End Sub
```

تخصيص أحد أزرار النموذج ليكون زر الالغاء

يمكن تخصيص أحد الأزرار على سطح النموذج، ليكون زر الإلغاء (Cancel Button). ويتم النقر على هذا الزر تلقائيا بمجرد أن يقوم المستخدم بالضغط على مفتاح ESC، بغض النظر عن أى متحكم أخر يحوز بؤرة التركيز على النموذج، ويتم برمجة هذا الزر للسماح بالمستخدم بالخروج السريع من الإجراء بدون تنفيذ أى عملية. ولتخصيص زر إلغاء باستخدام مصمم النماذج:

١. نفتح النموذج في مصمم النماذج.

٢. في نافذة Properties، نضبط خاصية CancelButton على اسم الزر المستهدف.

ولتخصيص زر إلغاء باستخدام الكود:

```
Private Sub SetCancelButton (ByVal myCancelBtn As Button)
  Me.CancelButton = myCancelBtn
End Sub
```

الإستجابة للنقر على زر بنماذج الويندوز

يستخدم متحكم Button في نماذج الويندوز أساسا، لتشغيل بعض الكود عند النقر على الزر. ويترتب على النقر على الزر أيضا مجموعة أخرى من الأحداث، تشمل MouseDown، MouseEnter، و MouseUp. وعند الرغبة في ربط هذه الأحداث بإجراءات معالجة، يجب التحقق من عدم وجود تناقض بين ما تقوم هذه الإجراءات بتنفيذه. ويجب الانتباه إلى أن متحكم الزر لا يدعم حدث النقر المزدوج، ولذلك يتم معالجة كل واحدة على حدة.

للاستجابة إلى عملية النقر على أحد الأزرار:

- في إجراء معالجة حدث النقر على الزر، ندخل الكود التالي:

```
Private Sub Button1_Click (ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
  MessageBox.Show ("Button1 was clicked")
End Sub
```

طرق اختيار متحكم زر على نموذج ويندوز

يمكن اختيار أحد الأزرار على نموذج الويندوز بإحدى الطرق التالية:

- النقر بالماوس على الزر
- استدعاء إجراء معالجة حدث النقر على الزر.
- نقل بؤرة التركيز إلى الزر باستخدام مفتاح TAB، ثم اختيار الزر بالضغط على قضيب المسافات أو مفتاح Enter.
- الضغط على مفتاح وصول خاص بالزر.
- إذا كان الزر هو الزر الإفتراضى على النموذج، يؤدى الضغط على مفتاح Enter إلى اختياره، حتى فى حالة حيازة متحكم أخر لبؤرة التركيز، إلا إذا كان المتحكم الأخر أحد الأزرار.
- ويؤدى الضغط على مفتاح ESC إلى اختيار الزر إذا كان هو زر الإلغاء، بغض النظر عن حيازة متحكم أخر لبؤرة التركيز.
- استدعاء وسيلة PerformClick لاختيار الزر باستخدام الكود، كما يتضح من الكود التالى.

```
Protected Sub button1_Click (sender As Object, e As EventArgs)
   Dim myVar as Integer
   If myVar Mod 2 = 0 Then
      button2.PerformClick ()
      MessageBox.Show ("button2 was clicked ")
   Else
      MessageBox.Show ("button2 was NOT clicked")
   End If
   myVar = myVar + 1
End Sub 'button1_Click
```

مربع الاختيار (CheckBox)

يشير مربع الاختيار بنماذج الويندوز إلى تحقق أحد الشروط أو عدم تحققة. ويستخدم بصفة عامة، لعرض اختيارات تنحصر فى نعم أو لا (Yes/No)، حقيقى أو غير حقيقى (True/False) أمام المستخدم. ويمكن استخدام مربعات الاختيار فى مجموعة لعرض اختيارات متعددة لكى يقوم المستخدم بالاختيار من بينها.

ويمكن ربط مربعات الاختيار مع عناصر فى قاعدة بيانات باستخدام الربط البسيط للبيانات. ويمكن وضع مجموعة من مربعات الاختيار فى مجموعة (GroupBox). ويحتوى مربع الاختيار على اثنين من الخصائص المهمة، خاصية الاختيار (Checked) وخاصية حالة الاختيار (CheckState). خاصية الاختيار تعيد إلينا القيمة True، التى تفيد بأن الاختيار قد تم، أو القيمة False التى تعيد تفيد عدم الاختيار. بينما تعيد خاصية CheckState القيمة CheckState.Checked أو القيمة CheckState.Unchecked.

الاستجابة للنقر على مربع الاختيار بنماذج الويندوز

عندما يقوم المستخدم بالنقر على مربع اختيار، يقع حدث النقر (Click Event). ويمكننا برمجة تطبيقاتنا للقيام بتنفيذ بعض الإجراءات بناءا على حالة مربع الاختيار. على سبيل المثال، نستخدم خاصية Checked لتحديد وضع المتحكم والاستجابة بناءا على هذا الوضع، كما يتضح من الكود التالى:

```
Private Sub CheckBox1_Click (ByVal sender As Object, ByVal e As System.EventArgs) Handles CheckBox1.Click
  If CheckBox1.Checked = True Then
    CheckBox1.Text = "Checked"
  Else
    CheckBox1.Text = "Unchecked"
  End If
End Sub
```

ويجب ملاحظة أن مربع الاختيار لا يدعم حدث النقر المزدوج، ولذلك يتم معالجة كل نقرة على حدة عند النقر المزدوج على هذا المتحكم. ولكي يتم اختيار المربع تلقائيا عند النقر علية، يجب ضبط خاصية AutoCheck على القيمة True، وهو الوضع الإفتراضى.

استخدام مربعات الاختيار لتحديد مجموعة من الأفعال التى يتم القيام بها

نستخدم عبارة Case للاستعلام عن قيمة خاصية CheckState لتقرير الأعمال التى يجب تنفيذها. وعندما يتم ضبط خاصية ThreeState على القيمة True، يمكن أن تعيد خاصية CheckState ثلاثة قيم ممكنة: القيمة Checked التى تعنى أن المربع تم اختياره، القيمة UnChecked التى تعنى أن المربع لم يتم اختياره، والقيمة Indeterminate التى تعنى أن المربع غير متاح للاستخدام. الكود التالى يوضح استخدام هذه الخاصية:

```
Private Sub CheckBox1_Click (ByVal sender As Object, ByVal e As
System.EventArgs) Handles CheckBox1.Click
      CheckBox1.ThreeState = True
      Select Case CheckBox1.CheckState
         Case CheckState.Checked
            CheckBox1.Text = "Box Checked"
         Case CheckState.Unchecked
            CheckBox1.Text = "Box UnChecked"
         Case CheckState.Indeterminate
            CheckBox1.Text = "Box Indeterminate"
      End Select
End Sub
```

ويجب الانتباه إلى أن ضبط خاصية ThreeState على القيمة True تؤدى إلى قيام خاصية Checked بإعادة القيمة True لكل من خاصية CheckState.Checked و خاصية CheckState.Indeterminate.

ضبط الخيارات باستخدام متحكم مربع الاختيار فى نماذج الويندوز

يستخدم مربع الاختيار فى معالجة نماذج الويندوز لتوفير خيارات True/False أو Yes/No أمام المستخدم. ويقوم هذا المتحكم بعرض علامة الاختيار عند اختياره.

لضبط الخيارات باستخدام مربعات الاختيار، نفحص قيمة خاصية Checked لتحديد حالتها، ثم استخدام هذه القيمة لضبط أحد الخيارات، كما يتضح من الكود التالى:

```
Private Sub SetOptions ()
   Dim MyObj As New myObject
   MyObj.Property1 = CheckBox1.Checked
   MyObj.Property2 = CheckBox2.Checked
End Sub
```

مربع سرد الاختيارات (CheckedListBox)

يقوم هذا المتحكم بعرض قائمة من البنود فى مربع وإتاحة إمكانية وضع علامة الاختيار بجانب البنود الموجودة فى تلك القائمة. ويعتبر ذلك المتحكم نسخة معدلة من متحكم مربع السرد (ListBox). ويقوم بجميع المهام التى يقوم بها مربع السرد بالإضافة إلى عرض علامة اختيار بجانب البنود الموجودة فى القائمة. كما أن مربع سرد الاختيارات يمكننا من إلقاء الضوء على بند واحد أو عدم إلقاء الضوء على أى بند. ويقبل مربع CheckedListBox الربط مع البيانات باستخدام خاصية DataSource.

معرفة البنود التى تم اختيارها فى مربع سرد الاختيارات

عند تقديم البيانات فى متحكم CheckedListBox بنماذج الويندوز، يمكننا تكرار اختيار مجموعة من البنود المخزنة فى خاصية CheckedItems، أو التنقل بين بنود القائمة باستخدام وسيلة GetItemChecked لمعرفة البنود التى تم اختيارها. ويتم تمرير رقم فهرس أحد البنود إلى وسيلة GetItemChecked. وتعيد هذه الوسيلة القيمة True أو القيمة False. ولا تحدد خاصية SelectedItems وخاصية SelectedIndices البنود التى تم اختيارها، بل تحددان البنود التى تم إلقاء الضوء عليها.

لتحديد البنود التى تم اختيارها فى هذا المتحكم، نتبع إحدى طريقتين:

- نكرر مراجعة البنود فى مجموعة CheckedItems، بدءا من المسلسل صفر لان بنود المجموعة تبدأ من الصفر. ويجب الإنتباة إلى هذه الوسيلة، تعيد إلينا رقم البند الذى تم اختياره فى قائمة البنود المختارة وليس فى كل القائمة. وعلى هذا الأساس، إذا كان البند الأول لم يتم فحصة وتم فحص البند الثانى، فإن الكود التالى سوف يعرض نص "Checked Item 1 = MyListItem2".

```
If CheckedListBox1.CheckedItems.Count <> 0 Then
  Dim X As Integer
  Dim S as String = ""
  For X = 0 To CheckedListBox1.CheckedItems.Count - 1
    S = S & "Checked Item " & (X + 1).ToString & " = " &
        CheckedListBox1.CheckedItems (X).ToString & ControlChars.CrLf
  Next X
  MessageBox.Show (S)
End If
```

• نستخدم وسيلة GetItemChecked مع كل بند من بنود القائمة، وسوف تعيد إلينا هذه الوسيلة رقم البند على مستوى كامل القائمة. وعلى هذا الأساس إذا كان البند الأول فى القائمة لم يتم فحصة وتم فحص البند الثانى، سوف يقوم الكود التالى بعرض نص "Item2 = MyListItem2".

```
Dim i As Integer
Dim s As String
s = "Checked Items:" & ControlChars.CrLf
For i = 0 to (CheckedListBox1.Items.Count - 1)
  If CheckedListBox1.GetItemChecked (i) = True Then
    s = s & "Item " & (i + 1).ToString & " = " & CheckedListBox1.Items(i).ToString
& ControlChars.CrLf
  End If
Next
MessageBox.Show (s)
```

مربع السرد المركب (ComboBox)

يستخدم متحكم ComboBox لعرض البيانات فى مربع سرد يتكون من جزأين. الجزء العلوي هو مربع النص الذى يسمح للمستخدم بإدخال بند قائمة السرد. والجزء الثانى عبارة عن مربع سرد يعرض قائمة منسدلة تحتوى على البنود التى يستطيع المستخدم الاختيار من بينها. فى هذا المتحكم، تعيد خاصية SelectedIndex قيمة صحيحة تمثل البند الذى تم اختياره فى قائمة السرد. ويمكن باستخدام الكود لتغيير البند المختار عن طريق تغيير قيمة هذه الخاصية. ويترتب على اختيار البند فى قائمة السرد، ظهوره فى الجزء الخاص بالنص فى مربع السرد المركب. وسوف تحتوى خاصية SelectedIndex على القيمة (-1) إذا لم يتم اختيار أى بند. وعند اختيار البند الأول فى القائمة، سوف تحتوى خاصية SelectedIndex على القيمة صفر. وتشابة خاصية SelectedItem خاصية SelectedIndex، ولكنها تعيد نص البند، الذى يكون فى صورة سلسلة حروف فى الغالب. ويمكن الحصول على عدد البنود فى قائمة السرد باستخدام خاصية Items.Count. وتكون قيمة هذه الخاصية دائما أكبر بواحد من القيمة الموجودة فى خاصية SelectedIndex لأن قيمة الخاصية الأخيرة تبدأ من الصفر. ويمكن الوصول إلى بنود قائمة السرد باستخدام خاصية Items، كما يتضح من الكود التالى:

```
Private Function GetItemText (i As Integer) As String
  Return ComboBox1.Items (i).ToString
End Function
```

ولإضافة أو حذف بنود فى متحكم ComboBox، نستخدم وسائل Items.Add، Items.Insert، Items.Remove، Items.Clear ويمكن أيضا إضافة بنود باستخدام خاصية Items فى نافذة Properties باستخدام مصمم النماذج. وتؤدى مربعات السرد المركبة إلى الاقتصاد فى المساحة المستخدمة على النموذج لأن القائمة لا يتم عرضها إلى أن يقوم المستخدم بالنقر على السهم المتجه إلى أسفل فى مربع النص. ولهذا يتناسب مربع السرد المركب مع المساحات الصغيرة المتاحة على النموذج. الأمثلة التالية توضح عمليات الإضافة والحذف باستخدام الوسائل السابقة:

- إضافة سلسلة حروف أو كائن باستخدام وسيلة Add:

```
ComboBox1.Items.Add ("Tokyo")
```

- إدراج سلسلة الحروف أو الكائن فى نقطة محددة فى القائمة باستخدام وسيلة Insert.

```
ComboBox1.Items.Insert (3, "Copenhagen")
```

- تخصيص مصفوفة كاملة لمجموعة Items:

```
Dim ItemObject (9) As System.Object
Dim i As Integer
  For i = 0 To 9
     ItemObject (i) = "Item" & i
  Next i
ComboBox1.Items.AddRange (ItemObject)
```

- حذف أحد البنود:

```
ComboBox1.Items.RemoveAt (0)
ComboBox1.Items.Remove (ComboBox1.SelectedItem)
ComboBox1.Items.Remove ("Tokyo")
```

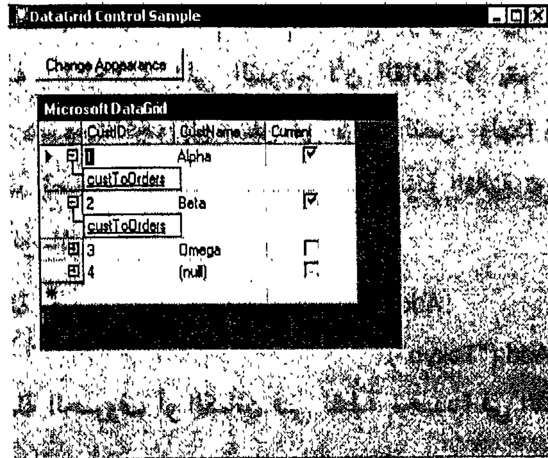
- حذف كل البنود:

```
ComboBox1.Items.Clear ()
```

شبكة البيانات (DataGrid)

يقدم متحكم شبكة البيانات واجهة بينية (Interface) خاصة بفئات بيانات ADO.NET عن طريق عرض البيانات مجدولة والسماح بتحديث مصدر هذه البيانات. وعند

ربط شبكة البيانات مع مصدر بيانات صحيح، يتم تأهيلها أتوماتيكيا بالبيانات التى تكون أعمدة وصفوف الشبكة. ويمكن استخدام هذا المتحكم لعرض جدول بيانات واحد أو عرض هيكل من البيانات يمثل العلاقات بين مجموعة من الجداول، كما يتضح من الشكل رقم (١١).



شكل رقم ١١

وعندما تكون شبكة البيانات مرتبطة مع عدد من الجداول التى بينها علاقات وتقبل فى نفس الوقت التجول داخلها، فإن الشبكة سوف تعرض عقد فى كل صف من صفوفها. تسمح العقد بالتنقل بين الجدول الأصلى وبين الجدول التابع ، حيث يؤدى النقر على إحدى العقد إلى عرض مربع سرد يحتوى على روابط مع الجداول التابعة مثل روابط الوب. والنقر على أيقونة "Show/Hide Parent Rows" يؤدى إلى إخفاء المعلومات الخاصة بالجدول الأصلى وإعادة النقر علية مرة أخرى يعكس الفعل.

ويعتبر عرض البيانات ومعالجتها وظيفتين مختلفتين. حيث تقوم شبكة البيانات بوظيفة التعامل مع واجهة الاستخدام ، بينما يقوم نظام ربط البيانات ومورد البيانات بمعالجة هذه البيانات. وعند ربط شبكة البيانات مع كائن فئة بيانات (DataSet)، يتم تلقائيا تكوين الصفوف والأعمدة وتشكيلها وتعبئتها بالبيانات. وبعد الانتهاء من تكوين شبكة البيانات، يمكن إضافة أعمدة، حذفها، أو إعادة صياغتها عند الحاجة.

ربط البيانات مع متحكم شبكة البيانات

لكى نستغل متحكم DataGrid، يجب ربطة مع مصدر بيانات باستخدام خاصية DataSource وخاصية DataMember فى وقت التصميم. كما يمكن القيام بنفس المهمة باستخدام وسيلة SetDataBinding فى وقت التشغيل. يؤدى ذلك إلى ربط شبكة البيانان مع كائن مصدر بيانات، مثل DataSet أو DataTable. يترتب على هذا الربط، قيام شبكة البيانات بعرض نتائج العمليات التى تنفذ على البيانات فى المصدر. ولا يتم تنفيذ معظم عمليات البيانات من خلال شبكة البيانات، ولكن من خلال مصدر البيانات. على سبيل المثال، إذا تم تحديث البيانات فى فئة البيانات، فإن شبكة البيانات تعكس هذه التغييرات. وعند ضبط خاصية ReadOnly على قيمة False فى شبكة البيانات والأعمدة والصفوف المرتبطة بها، يمكن استخدام الشبكة فى تحديث فئة البيانات المرتبطة بها. وعند تعريف علاقة الأصل والتابع بين جداول البيانات، فإن المستخدم يمكنة التنقل بين الجداول ذات العلاقة لانتقاء الجدول الذى يريد عرضة.

مصادر البيانات الصالحة للارتباط بشبكة البيانات

لقد تم تصميم شبكة البيانات خصيصا لعرض المعلومات الموجودة فى مصدر بيانات. ويمكن تحديد المصادر الصالحة للربط مع شبكة البيانات فى المصادر التالية :

- تصنيف جدول بيانات (DataTable Class)
- تصنيف مشهد بيانات (DataView Class)
- تصنيف فئة بيانات (DataSet Class)
- تصنيف مدير مشاهد بيانات (DataView Manager Class)

ونقوم بربط الشبكة وقت التصميم عن طريق ضبط خاصية DataSource وخاصية DataMember، أو فى وقت التشغيل عن طريق استدعاء وسيلة SetDataBinding. ومع أنة يمكن عرض البيانات من مصادر متعددة، إلا أن المصادر الرئيسية المستخدمة هى فئات البيانات (Datasets) ومشاهد البيانات (Data Views). وعند ربط فئة بيانات مع شبكة بيانات، يمكن أن تكون أحد الكائنات الموجودة على النموذج أو كائن يتم تمريرة إلى النموذج بواسطة تطبيق XML Web Service. عندما يقوم متحكم شبكة البيانات بعرض

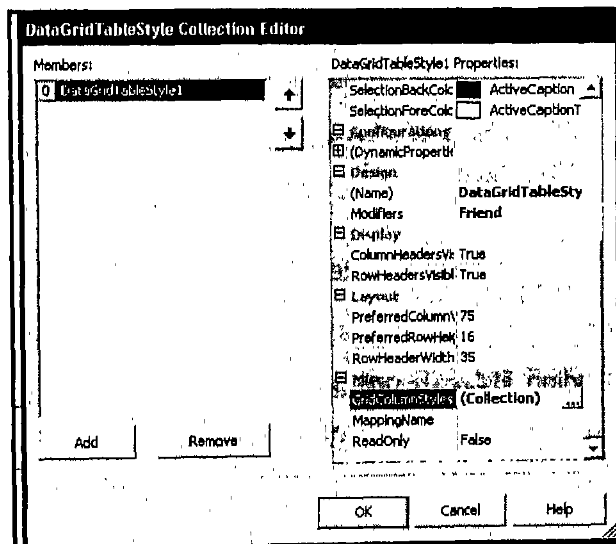
جدول بيانات مع ضبط خاصية AllowSorting على القيمة True، يمكن إعادة فرز البيانات بالنقر على رأس العمود. كما يمكن للمستخدم أيضا إضافة الصفوف وتحرير الخلايا.

أعمدة وصفوف شبكة البيانات

يمكن تقسيم الكائنات التى تحتوى عليها شبكة البيانات (DataGrid) على النحو التالى :

- يحتوى كل كائن DataGrid على مجموعة من كائنات DataGridTableStyle التى تحددها خاصية TableStyles.
- يحتوى كل كائن DataGridTableStyle على مجموعة من كائنات DataGridColumnStyle التى تحددها خاصية GridColumnStyles.

ويمكن تدقيق خاصية TableStyles وخاصية GridColumnStyles باستخدام محرر المجموعات الذى يمكن الوصول إلية باستخدام نافذة Properties. يمكن الوصول إلى كائنات DataGridTableStyle المرتبطة مع متحكم DataGrid من خلال كائن GridTableStylesCollection. ويمكن تدقيق تلك المجموعة فى المصمم باستخدام محرر DataGridTableStyle Collection، الموضح الشكل رقم (١٢). كما يمكن كتابة الكود الذى يستخدم خاصية TableStyles مباشرة.



شكل رقم ١٢

تحقيق التزامن بين مصدر البيانات وبين شبكة البيانات

تتزامن أنماط الجداول (Table Styles) وأنماط الأعمدة (Column Styles) مع كائنات جداول البيانات (DataTable Objects) وكائنات أعمدة البيانات (DataColumn Objects) عن طريق ضبط خاصية MappingName على قيمة خاصية TableName وقيمة خاصية ColumnName المناسبة. وعند إضافة كائن DataGridTableStyle إلى متحكم DataGrid مرتبط مع مصدر بيانات صالح، ثم ضبط خاصية MappingName مع خاصية TableName صالحة، يتم إضافة كائن DataGridColumnStyle مقابل كل كائن DataColumn في مجموعة Columns بكائن جدول البيانات.

إضافة جداول وأعمدة إلى شبكة بيانات بنموذج ويندوز

يمكن عرض البيانات على نموذج الويندوز فى شبكة تحتوى على مجموعة من الصفوف والأعمدة عن طريق إضافة كائنات DataGridTableStyle إلى كائن GridTableStylesCollection، التى يمكن الوصول إليها من خلال خاصية TableStyles فى كائن DataGrid. ويعرض كل نمط من أنماط جداول شبكة البيانات، الجدول الذى يتم تحديد فى خاصية MappingName بكائن DataGridTableStyle. فى الوضع الإفتراضى، يعرض نمط جدول البيانات الذى لا يحتوى على أنماط أعمدة، جميع الأعمدة فى جدول البيانات. ويمكن تقييد أعمدة الجدول المعروضة بإضافة كائنات DataGridColumnStyle إلى كائن GridColumnStylesCollection، التى يمكن الوصول إليها باستخدام خاصية GridColumnStyles فى كائن DataGridTableStyle.

لإضافة جدول إلى شبكة البيانات أثناء التصميم:

١. نقوم بربط متحكم DataGrid مع فئة بيانات (DataSet).

٢. نختار خاصية TableStyles بمتحكم DataGrid فى نافذة Properties، ثم ننقر نقاط الحذف التالية للخاصية لعرض نافذة DataGridTableStyle Collection Editor.

٣. فى نافذة Collection Editor، ننقر زر Add لإضافة نمط جدول.

٤. ننقر Ok لإقفال نافذة Collection Editor، ثم نعيد فتحة بالنقر على نقاط الحذف

التالية لخاصية TableStyles. يترتب على ذلك، عرض جداول البيانات المرتبطة مع الشبكة فى القائمة المنسدلة بخاصية MappingName بكائن TableStyle.

٥. فى مربع Members بنافذة DataGridTableStyle Collection Editor، نختار نمط الجدول.

٦. فى مربع Properties بمحرر المجموعة، نختار جدول الربط مع خاصية MappingName.

لإضافة أحد الأعمدة لشبكة بيانات أثناء التصميم:

١. فى مربع Members بنافذة DataGridTableStyle Collection Editor نختار نمط الجدول.

٢. فى نافذة Properties، نختار GridColumnStyles ثم ننقر نقاط الحذف التالية للخاصية لعرض نافذة DataGridColumnStyle Collection Editor.

٣. فى نافذة محرر المجموعة، ننقر زر Add لإدراج نمط عمود أو ننقر السهم التالى لزر Add لتحديد نمط العمود. سوف تمكننا القائمة المنسدلة من اختيار أحد نوعين: DataGridTextBoxColoumn أو DataGridBoolColumn.

٤. ننقر Ok لإقفال نافذة DataGridColumnStyle Collection Editor، ثم نعيد فتحها بالنقر على نقاط الحذف التالية لخاصية DataGridColumnStyle Collection Editor . عند إعادة فتح محرر المجموعة، سوف تظهر الأعمدة التى يحتوى عليها الجدول المرتبط فى القائمة المنسدلة فى خاصية MappingName بنمط العمود.

٥. فى مربع Members بمحرر المجموعة، ننقر على نمط العمود.

٦. فى مربع Properties بمحرر الكود، نختار قيمة MappingName الخاصة بالعمود الذى نريد عرضة.

لإضافة جدول وعمود إلى شبكة بيانات باستخدام الكود:

١. نقوم أولا بربط متحكم DataGrid مع كائن DataSet.

٢. نعلن عن نمط جدول جديد ونضبط خاصية MappingName باستخدام الكود التالى:

```
Dim ts1 as New DataGridTableStyle ()
ts1.MappingName = "Customers"
```

٣. نعلن عن نمط عمود جديد ونضبط خاصية MappingName والخصائص الأخرى باستخدام الكود التالى:

```
Dim myDataCol As New DataGridBoolColumn ()
myDataCol.HeaderText = "My New Column"
myDataCol.MappingName = "Current"
```

٤. نستدعى وسيلة Add فى كائن GridColumnStylesCollection لإضافة العمود إلى نمط الجدول، كما يتضح من الكود التالى:

```
ts1.GridColumnStyles.Add (myDataCol)
```

٥. نستدعى وسيلة Add فى كائن GridTableStyleCollection لإضافة نمط جدول إلى شبكة البيانات. كما يتضح من الكود التالى:

```
DataGrid1.TableStyles.Add (ts1)
```

إظهار شبكة البيانات

الاستخدام الشائع لشبكة البيانات هو عرض بيانات جدول واحد من بين جداول فئة البيانات. من ناحية أخرى، يمكن استخدام هذا المتحكم فى عرض جداول متعددة، بما فى ذلك الجداول التى بينها علاقات. ويجرى تهيئة شبكة العرض تلقائيا طبقا لمصدر البيانات:

- عندما يكون مصدر البيانات جدولا، يتم العرض فى شبكة.
- وعندما يكون مصدر البيانات جداول متعددة، يمكن أن تعرض الشبكة مشهد شجرة يقوم المستخدم بالتنقل بين أجزائها للوصول إلى الجدول الذى يريد عرضة.
- وعندما يكون مصدر البيانات جداول متعددة بينها علاقات، يمكن أن تعرض الشبكة شجرة لاختيار أحد الجداول منها. كما يمكن عرض الجدول الأصلى واستخدام سجلاته للوصول إلى السجلات التابعة.

ربط شبكة البيانات

إذا كان مصدر البيانات متاحا فى وقت التصميم، يمكننا ربط شبكة البيانات مع مصدرها وقت التصميم. ويمكن فى هذه الحالة المشاهدة المبدئية لما سوف تظهر علية

البيانات فى الشبكة. ويمكن أيضا استخدام الكود فى ربط شبكة البيانات وقت التشغيل. على سبيل المثال، يمكن اختيار اسم جدول أو مشهد وقت التشغيل. ويعتبر ذلك ضروريا فى الحالات التى لا تتواجد فيها مصادر البيانات أثناء التصميم.

لربط شبكة البيانات مع جدول منفرد فى وقت التصميم:

١. نضبط خاصية DataSource بشبكة البيانات مع الكائن الذى يحتوى على البنود المطلوبة.

٢. إذا كان مصدر البيانات فئة بيانات، نضبط خاصية DataMember على الجدول الذى يتم الربط معه.

٣. إذا كان مصدر البيانات فئة بيانات أو مشهد يستخدم فئة البيانات، نضيف كود إلى النموذج لتأهيل الفئة ببيانات مصدر البيانات. ويعتمد الكود المستخدم على المصدر الذى تحصل منة فئة البيانات على بياناتها. عند تأهيل فئة البيانات مباشرة من قاعدة البيانات، نستدعى وسيلة Fill فى موفق البيانات (DataAdapter)، كما فى المثال التالى الذي يؤهل فئة بيانات باسم dsCategories1:

```
sqlDataAdapter1.Fill (DsCategories1)
```

ويمكن أن نضيف أنماط الجداول والأعمدة المناسبة اختياريا إلى شبكة البيانات. وإذا لم يكن هناك أنماط جداول، سوف نرى محتويات الجدول مع أقل قدر من التشكيل، كما يتم عرض جميع الأعمدة.

لربط شبكة البيانات مع عدة جداول فى فئة بيانات فى وقت التصميم:

١. نضبط خاصية DataSource على الكائن الذى يحتوى على بنود البيانات التى نريد عرضها.

٢. إذا كانت فئة البيانات تحتوى على جداول بينها علاقات، نضبط خاصية DataMember على اسم الجدول الأصلى.

٣. نكتب الكود اللازم لتأهيل فئة البيانات.

لربط شبكة البيانات باستخدام الكود:

١. نكتب الكود اللازم لتأهيل فئة البيانات.

٢. نستدعى وسيلة SetDataBinding بكائن فئة البيانات، مع تمرير مصدر وعضو البيانات إلية. وإذا لم تكن هناك حاجة إلى تمرير عضو بيانات صراحة إلى الوسيلة، نقوم بتمرير سلسلة حروف خالية. عندما تحتوى فئة البيانات على أكثر من جدول، يمكننا استخدام الكود التالى لتحقيق الربط:

```
DataGrid1.SetDataBinding (DsCustomers1, "Customers")
```

وإذا كان الجدول هو الجدول الوحيد فى فئة البيانات، يمكننا ربط الشبكة باستخدام الكود التالى:

```
DataGrid1.SetDataBinding (DsCustomers1, "")
```

٣. نضيف أنماط الجداول والأعمدة المناسبة اختياريا. فإذا لم يكن هناك أنماط جداول، سوف يتم عرض جميع حقول جدول البيانات مع أقل قدر من الصياغة.

تغيير البيانات المعروضة فى شبكة البيانات وقت التشغيل

بعد الانتهاء من تكوين شبكة البيانات فى وقت التصميم، قد نرغب فى تغيير عناصر كائن DataSet المرتبط بالشبكة ديناميكيا فى وقت التشغيل. يمكن أن يشمل ذلك تغيير القيم الفردية فى الجدول أو تغيير مصدر البيانات المرتبط بشبكة البيانات. تغيير القيم الفردية فى الجدول يتم من خلال كائن فئة البيانات (DataSet) وليس من خلال شبكة البيانات (DataGrid).

لتغيير البيانات فى وقت التشغيل باستخدام الكود:

- نحدد الجدول المطلوب فى كائن فئة البيانات (DataSet)، وكذلك الصف والحقل فى ذلك الجدول ثم نضبط الحقل على القيمة الجديدة. ويتم تحديد الجدول الأول فى فئة البيانات أو الصف الأول فى الجدول باستخدام القيمة صفر. يبين المثال التالى كيفية تغيير الحقل الثانى فى الصف الأول فى فئة بيانات بالنقر على زر Button1. فى هذا المثال، نفترض تكوين فئة البيانات ds والجداول أرقام 0، و 1 فى وقت سابق.

```
Protected Sub Button1_Click (ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
```

```
  ds.tables (0).rows(0)(1)="NewEntry"
End Sub
```

ويمكن استخدام خاصية SetDataBinings في وقت التشغيل، لربط متحكم فئة بيانات مع مصدر بيانات مختلف. على سبيل المثال، يمكن أن يكون لدينا عدد من أدوات تحكم ADO.NET التي يتصل كل واحد منها مع قاعدة بيانات مختلفة.

لتغيير مصدر البيانات باستخدام الكود:

- نستخدم وسيلة SetDataBinding للربط مع مصدر البيانات والجدول المستهدف، كما يتضح من المثال التالى الذى يستخدم موفق بيانات مع جدول Authors في قاعدة بيانات Pupbs.

```
Private Sub ResetSource ()
  DataGrid1.SetDataBinding (adoPubsAuthors, "Authors")
End Sub
```

تكوين علاقات الأصل والتابع باستخدام متحكم DataGrid

عندما تحتوى فئة البيانات على سلسلة من الجداول ذات العلاقات، يمكننا استخدام اثنين من أدوات DataGrid لعرض البيانات في صيغة الأصل والتابع (Master-Detail). حيث يتم تخصيص إحدى كائنات DataGrid لتكون الشبكة الأساسية، تخصيص كائن DataGrid أخر ليكون الشبكة التفصيلية للبيانات. وعند اختيار أحد الإدخالات في القائمة الأصلية، يتم عرض كل الإدخالات المرتبطة به في الشبكة التفصيلية. على سبيل المثال، إذا كان هناك فئة بيانات تحتوى على جدول عملاء (Customer Table) وعلى جدول أوامر المبيعات الخاصة بهؤلاء العملاء، سوف نفترض أن جدول العملاء هو الجدول الذى يعرض في كائن DataGrid الأصلى وجدول الأوامر هو الجدول الذى سوف يعرض في كائن DataGrid التابع.

حذف أعمدة من متحكم DataGrid

يمكن حذف الأعمدة باستخدام الكود من متحكم شبكة البيانات (DataGrid) عن طريق استخدام الخصائص والوسائل الموجودة في كائن GridColumnStylesCollection وكائن DataGridColumnStyle.

لحذف عمود من شبكة بيانات باستخدام الكود:

نستدعى وسيلة RemoveAt بكائن GridColumnStylesCollection مع تحديد رقم العمود المطلوب حذفة. كما يتضح من الكود التالى:

```
Dim ColumnIndex As Integer = 1
Dim myGridColumns As GridColumnStylesCollection
myGridColumns = DataGrid1.TableStyles (0).GridColumnStyles
myGridColumns.RemoveAt (ColumnIndex)
```

صياغة متحكم DataGrid فى وقت التصميم

يمكن صياغة خواص متحكم DataGrid، مثل الألوان، طاقم الحروف، الألوان الخلفية، وخصائص البيانات أثناء التصميم باستخدام مربع Properties. ويمكن أيضا صياغة شكل عرض الشبكات عن طريق تكوين كائنات أنماط جداول شبكة البيانات (DataGridTableStyle) وإضافتها إلى مجموعة (GridTableStylesCollection). ولتعديل شكل ظهور الأعمدة الفردية بما يتفق مع رغبات المستخدم، نضيف كائنات نمط عمود شبكة البيانات (DataGridColumnStyle) إلى كائن مجموعة أنماط أعمدة الشبكة (GridColumnStylesCollection) لكل نمط من أنماط الجداول السابق إضافتها. وهناك نوعان من أنماط الأعمدة التى توفر إمكانيات كبيرة فى الصياغة، هى: نمط DataGridBoolColumn الذى يقبل ويعرض قيم True، False، Null. والنمط الأخر يقبل ويعرض البيانات فى شكل سلاسل، وهو DataGridTextBoxColumn. وإمكانيات التحرير التى يتيحها هذا الكائن هى نفسها إمكانيات التحرير الموجودة فى كائن TextBox. ولصياغة البيانات التى يتم عرضها، نضبط خاصية Format فى كائن DataGridTextBoxColumn على إحدى قيم الصياغة.

الاستجابة للنقر بزر الماوس فى متحكم DataGrid

بعد تنفيذ اتصال بين شبكة بيانات (DataGrid) وبين قاعدة بيانات (Database)، يمكننا مراقبة الخلية التى يقوم المستخدم بالنقر عليها فى شبكة البيانات.

للتحرى عن الوقت الذى يقوم فيه المستخدم باختيار خلية مختلفة:

• ندخل الكود التالى فى إجراء معالجة حدث CurrentCellChanged الذى يقع عند

تغيير الخلية.

```
Private Sub myDataGrid_CurrentCellChanged (ByVal sender As Object, ByVal e As
System.EventArgs) Handles myDataGrid.CurrentCellChanged
  MessageBox.Show ("Col is " & myDataGrid.CurrentCell.ColumnNumber _
    & ", Row is " & myDataGrid.CurrentCell.RowNumber _
    & ", Value is " & myDataGrid.Item (myDataGrid.CurrentCell))
End Sub
```

- ولمعرفة جزء الشبكة الذى قام المستخدم بالنقر علية، نستدعى وسيلة HitTest فى إجراء معالجة أحد الأحداث المناسبة، مثل حدث Mousedown أو حدث Click. تعيد وسيلة HitTest كائن يحتوى على الصف والعمود الذى تم النقر عليهما، وهو كائن DataGridHitTestInfo.

```
Private Sub myDataGrid_MouseDown (ByVal sender As Object, _
ByVal e As MouseEventArgs) Handles myDataGrid.MouseDown
  Dim myGrid As DataGrid = CType (sender, DataGrid)
  Dim hti As System.Windows.Forms.DataGrid.HitTestInfo
  hti = myGrid.HitTest(e.X, e.Y)
  Dim message As String = "You clicked "

  Select Case hti.Type
    Case System.Windows.Forms.DataGrid.HitTestType.None
      message &= "the background."
    Case System.Windows.Forms.DataGrid.HitTestType.Cell
      message &= "cell at row " & hti.Row & ", col " & hti.Column
    Case System.Windows.Forms.DataGrid.HitTestType.ColumnHeader
      message &= "the column header for column " & hti.Column
    Case System.Windows.Forms.DataGrid.HitTestType.RowHeader
      message &= "the row header for row " & hti.Row
    Case System.Windows.Forms.DataGrid.HitTestType.ColumnResize
      message &= "the column resizer for column " & hti.Column
    Case System.Windows.Forms.DataGrid.HitTestType.RowResize
      message &= "the row resizer for row " & hti.Row
    Case System.Windows.Forms.DataGrid.HitTestType.Caption
      message &= "the caption"
    Case System.Windows.Forms.DataGrid.HitTestType.ParentRows
      message &= "the parent row"
  End Select
  Console.WriteLine (message)
```

```
End Sub
```

التحقق من صحة الإدخالات في متحكم DataGrid

هناك نوعان من تدقيق الإدخالات في شبكة البيانات. عندما يحاول المستخدم إدخال قيمة من نوع غير مقبول في خلية، على سبيل المثال إدخال سلسلة في موقع رقم صحيح، يتم إحلال القيمة غير الصحيحة بالقيمة السابقة. وهذا النوع من الإدخالات، يتم تدقيقة تلقائيا. النوع الثانى من تدقيق الإدخالات يمكن استخدامه لرفض أى بيانات غير مقبولة، على سبيل المثال، القيمة صفر فى حقل يقبل القيم التى تساوى قيمة أكبر من الصفر أو سلسلة حروف غير مناسبة. للقيام بهذا النوع من التدقيق، نستخدم إجراء معالجة حدث DataTableColumnChanging الخاص بتغيير عمود جدول بيانات، أو حدث RowChanging الخاص بتغيير صف فى جدول بيانات.

المثال التالى يستخدم حدث ColumnChanging لأن القيمة غير الصحيحة تقع فى عمود "Product". كما يمكن استخدام حدث RowChanging لتدقيق أن القيمة فى عمود "EndDate" تكون بعد القيمة الموجودة فى عمود "StartDate" بنفس الصف.

١. نكتب الكود اللازم لمعالجة حدث ColumnChanging وعند اكتشاف أخطاء نستدعى وسيلة SetColumnError.

```
Private Sub Customers_ColumnChanging (ByVal sender As Object, _
ByVal e As System.Data.DataColumnChangeEventArgs)
  ' Only check for errors in the Product column
  If (e.Column.ColumnName.Equals ("Product")) Then
    ' Do not allow "Automobile" as a product.
    If CType (e.ProposedValue, String) = "Automobile" Then
      Dim badValue As Object = e.ProposedValue
      e.ProposedValue = "Bad Data"
      e.Row.RowError = "The Product column contians an error"
      e.Row.SetColumnError(e.Column, "Product cannot be " & _
      CType (badValue, String))
    End If
  End If
End Sub
```

٢. نربط إجراء Customers_ColumnChanging مع حدث ColumnChanging.

```
AddHandler customersDataSet1.Tables ("Customers").ColumnChanging, AddressOf
Customers_ColumnChanging
```

مربع اختيار التاريخ والوقت (DateTimePicker)

يسمح لنا متحكم DateTimePicker باختيار أحد البنود من قائمة سرد تواريخ أو أوقات. ويتكون شكل هذا المتحكم من جزأين: الجزء الأول عبارة عن مربع نص على يمينه سهم متجه إلى الأسفل يحتوى على التواريخ أو الأوقات فى صورة نصوص، والجزء الثانى عبارة عن قائمة سرد تظهر عند النقر على السهم الموجود بمربع النص، ويمكن اختيار التواريخ الأوقات من هذه القائمة. ويمكن ضبط خاصية ShowUpDown بمتحكم DateTimePicker على القيمة True لإظهار أزرار Up و Down، التى يمكن استخدامها فى تغيير التواريخ. كما يمكن ضبط خاصية ShowCheckBox على القيمة True لعرض مربع اختيار بجانب التاريخ الذى يتم اختياره بالمتحكم. وعند وضع علامة الاختيار فى المربع، يمكننا تحديث التاريخ، بينما يترتب على حذفها حجب إمكانية تغيير القيمة.

وتحدد خاصية MaxDate التاريخ الأقصى وخاصية MinDate التاريخ الأدنى فى نطاق التواريخ والأوقات التى يمكن استخدامها. وتحتوى خاصية Value على التاريخ والوقت الجاري الذى تم اختياره من المتحكم. ويمكن عرض القيم فى أربعة صيغ عن طريق ضبط خاصية Format. تتمثل هذه القيم الأربعة فى : الصيغة الطويلة، الصيغة القصيرة، صيغة الوقت، والصيغة المعدلة. إذا تم اختيار الصيغة المعدلة، يجب ضبط خاصية تعديل الصيغة طبقا لاختيارات المستخدم.

عرض التاريخ فى صيغة معدلة

يوفر لنا متحكم DateTimePicker المرونة اللازمة لإعادة صياغة عرض التواريخ والأوقات. حيث تسمح لنا خاصية Format فى هذا المتحكم بالاختيار بين الصيغ السابق إعدادها، الصيغ الموجودة فى تعداد صيغ التواريخ والوقت (DateTimeFormat Enumeration). وعندما لا تكن هذه الصيغ كافية، يمكننا تكوين الصيغ الخاصة بنا باستخدام حروف الصيغ التى يتم سردها فى خاصية CustomFormat.

لعرض صيغة من الصيغ التى يحددها المستخدم:

١. نضبط خاصية Format على قيمة DateTemePickerFormat.Custom.

٢. نضبط خاصية CustomFormat على سلسلة تمثل الصيغة التى يحددها المستخدم، كما يتضح من الكود التالى بيانه:

```
DateTimePicker1.Format = DateTimePickerFormat.Custom
DateTimePicker1.CustomFormat = "ddd dd MMM yyyy"
```

إضافة نص إلى صيغة التاريخ والوقت:

يمكننا إضافة نص إلى سلسلة الصيغة عن طريق استخدام علامات التنصيص المنفردة حول النص. وبصفة عامة، يجب وضع علامات التنصيص المنفردة حول أى رمز ليس من رموز الصيغة. على سبيل المثال، الصيغة التالية تعرض التاريخ الحالى فى صيغة " Today is : 05:30:31 Friday March 03, 2001" باستخدام اللغة الإنجليزية.

```
DateTimePicker1.CustomFormat = "'Today is:' hh:mm:ss dddd MMMM dd, yyyy"
```

استخدام DateTimePicker لإعادة التواريخ

تتحكم خاصية Value فى التاريخ أو الوقت الذى يتم اختياره فى متحكم DateTimePicker. ويمكن ضبط قيمة هذه الخاصية قبل عرض المتحكم. على سبيل المثال، فى وقت التصميم أو فى إجراء معالجة حدث Form_Load، يمكننا ضبط التاريخ الذى يتم البدء به عند تشغيل DateTimePicker بدلا من استخدام التاريخ الحالى افتراضيا. ويمكن تغيير هذه القيمة باستخدام الكود لكى يعكس المتحكم القيمة الجديدة أتوماتيكيا أثناء التشغيل. وتعيد خاصية Value هيكل بيانات يحتوى على التاريخ والوقت. ويحتوى هذا الهيكل على عدد من الخصائص التى توفر معلومات خاصة عن التاريخ والوقت المعروضين. ويمكن قراءة هذه الخصائص فقط.

- بالنسبة للتاريخ، تشتمل هذه الخصائص على خاصية Month, Day, Year التى تعيد قيما صحيحة تمثل وحدات التاريخ الذى يتم اختياره. وتعيد خاصية DayOfWeek قيمة تشير إلى اليوم الذى يتم اختياره فى الأسبوع.
- وبالنسبة للوقت، تشتمل خصائص هيكل البيانات التى تعيده خاصية Value على خاصية Hour، خاصية Minute، خاصية Second، وخاصية MilliSecond التى تحتوى على قيم صحية تمثل أجزاء الوقت.

لضبط التاريخ أو الوقت:

```
DateTimePicker1.Value = New DateTime (2001, 10, 20)
```

لإعادة التاريخ أو الوقت:

هناك خياران، الأول هو استخدام خاصية Text لإعادة كامل قيمة التاريخ والوقت التى تم اختيارها، والثاني هو استدعاء الوسيلة المناسبة فى خاصية Value لإعادة جزء من القيمة. ونستخدم ToString لتحويل المعلومات إلى سلسلة يمكن عرضها أمام المستخدم. الكود التالى يوضح ذلك:

```
MessageBox.Show ("The selected value is ", DateTimePicker1.Text)
MessageBox.Show ("The day of the week is ",
DateTimePicker1.Value.DayOfWeek.ToString)
MessageBox.Show ("Millisecond is: ", dateTimePicker1.Value.Millisecond.ToString)
```

مدرج النطاق (DomainUpDown)

يماثل متحكم DomainUpDown مركب من كائن TextBox وزوجين من الأسهم للحركة إلى أعلى والى أسفل خلال قائمة سرد. ويعرض هذا المتحكم ويضبط سلسلة نص من بين قائمة من الخيارات. ويمكن للمستخدم أن يختار السلسلة بإحدى الطرق التالية:

- النقر على زر السهم المتجه إلى أعلى أو زر السهم المتجه إلى اسفل
- الضغط على مفتاح UP أو DOWN
- طباعة سلسلة مماثلة لأحد السلاسل فى القائمة.

واحد الاستخدامات الممكنة لهذا المتحكم هو اختيار بنود من قائمة مفروزة من الأسماء. ولكى نفرز هذه القائمة، نضبط خاصية Sorted على القيمة True. وتشابة وظيفة هذا المتحكم إلى حد كبير، وظيفة متحكم ListBox ومتحكم ComboBox، ولكنة يتميز بشغله مساحة صغيرة جدا فى النموذج.

وأهم خصائص هذا المتحكم هى خاصية Items، خاصية ReadOnly، وخاصية Wrap. تحتوى Items على قائمة بالكائنات التى يتم عرض قيمة خاصية Text بها فى المتحكم. وعند ضبط خاصية ReadOnly على القيمة False، يقوم المتحكم أتوماتيكيا باستكمال النص الذى يدخلة المستخدم بالمقارنة مع قيمة فى القائمة. ويترتب على ضبط خاصية Wrap على

القيمة True، العودة إلى البند الأول فى القائمة عند تجاوز البند الأخير فى الاختيارات، والعكس صحيح. وأهم الوسائل التى يشتمل عليها هذا المتحكم هى: وسيلة UpButton، وسيلة DownButton. ويتميز هذا المتحكم بأنة يعرض سلاسل من النصوص فقط. وعند الرغبة فى عرض أرقام، يمكن استخدام متحكم NumericUpDown.

إضافة وحذف بنود إلى متحكم DomainUpDown

يمكننا إضافة بنود إلى هذا المتحكم باستخدام الكود. يتم ذلك باستدعاء وسيلة Add أو وسيلة Insert فى تصنيف DomainUpDownItemCollection لإضافة بنود إلى خاصية Items بهذا المتحكم. تستخدم وسيلة Add فى الإضافة بنهاية القائمة، بينما تستخدم وسيلة Insert لإضافة بند فى موقع محدد بالقائمة.

لإضافة بند جديد، نستخدم الكود التالى:

```
DomainUpDown1.Items.Add ("noodles")
```

لإدراج بند فى موقع محدد بالقائمة، نستخدم الكود التالى:

```
DomainUpDown1.Items.Insert (2, "rice")
```

ويمكننا أيضا حذف بند من متحكم DomainUpDown باستخدام اسم البند، كما يتضح من الكود التالى:

```
DomainUpDown1.Items.Remove ("noodles")
```

كما يمكن حذف أحد البنود باستخدام موقعة فى الفهرس، كما يتضح من الكود التالى:

```
DomainUpDown1.Items.RemoveAt (0)
```

يقوم المثال التالى باستنساخ وإعداد متحكم DomainUpDown. كما يقوم بضبط بعض خصائص المتحكم ويكون مجموعة من السلاسل للعرض فى متحكم DomainUpDown. ويفترض الكود وجود متحكم TextBox، متحكم CheckBox، ومتحكم Button على النموذج. ويمكن إدخال سلسلة فى مربع النص وإضافتها إلى مجموعة Items عند النقر على متحكم Button. وبالنقر على مربع الاختيار، يمكن التحول بين الفرز وعدم الفرز للبنود ومشاهدة تأثير ذلك على مجموعة متحكم DomainUpDown.

```
 Protected domainUpDown1 As DomainUpDown
Private Sub MySub ()
   domainUpDown1 = New System.Windows.Forms.DomainUpDown ()
```

```
   Controls.Add (domainUpDown1)
End Sub
Private Sub button1_Click (sender As System.Object, e As System.EventArgs)
   domainUpDown1.Items.Add ((textBox1.Text.Trim() & " - " & myCounter))
   myCounter = myCounter + 1
   textBox1.Text = ""
End Sub
Private Sub checkBox1_Click (sender As System.Object, e As System.EventArgs)
   If domainUpDown1.Sorted Then
      domainUpDown1.Sorted = False
   Else
      domainUpDown1.Sorted = True
   End If
End Sub
Private Sub domainUpDown1_SelectedItemChanged _
   (sender As System.Object, e As System.EventArgs)
      MessageBox.Show (("SelectedIndex: " &
domainUpDown1.SelectedIndex.ToString () & _
  ControlChars.Cr & "SelectedItem: " & domainUpDown1.SelectedItem.ToString
()))
End Sub 'domainUpDown1_SelectedItemChanged
```

مربع المجموعة (GroupBox)

يستخدم متحكم GroupBox لتوفير وضع أدوات التحكم الأخرى في مجموعات. وتستخدم GroupBox بصفة أساسية لتقسيم النموذج إلى أقسام فرعية على أساس وظيفي. على سبيل المثال، يمكن أن يكون لدينا نموذج أمر مبيعات (Order Form) يحدد خيارات البريد المستخدم في إرسال البضاعة إلى العملاء. ويؤدى وضع أدوات التحكم في مجموعات، إلى توفر إمكانية تحريك مجموعات الأدوات بسهولة في وقت التصميم عن طريق تحريك المربع الواقعة به. ويحتوى متحكم GroupBox على عنوان يمكن تعريفة في خاصية Text.

لتكوين مجموعة من أدوات التحكم:

١. نرسم متحكم GroupBox على سطح النموذج.

٢. نضيف أدوات تحكم أخرى إلى مربع المجموعة. يتم ذلك بسحب كل متحكم إلى داخل المجموعة. وإذا كان لدينا مجموعة من أدوات التحكم على النموذج ونريد

وضعها فى مجموعة، نختار هذه الأدوات ثم نقصها ونلصقها فى مربع المجموعة. ويمكن أيضا سحبها إلى مربع المجموعة.

٣. نضبط خاصية Text فى مربع المجموعة على العنوان المناسب.

المميز (Label)

تستخدم هذه الأداة لعرض نص أو صورة لايمكن تغييرها بواسطة المستخدم. وتستخدم هذه الأدوات لتعريف كائنات على نموذج، لتوفير وصف لما سيقوم به متحكم من نوع معين، أو عرض معلومات استجابة لأحد الأحداث أو العمليات أثناء التشغيل. على سبيل المثال، يمكن استخدام متحكم Label لإضافة عناوين إلى مربعات النصوص (Text Boxes)، مربعات السرد (ListBoxes)، مربعات السرد المركبة (ComboBoxes)، وغيرها من أدوات التحكم. ويمكن أيضا كتابة كود يقوم بتغيير النص المعروض بواسطة متحكم Label استجابة لأحداث وقت التشغيل. على سبيل المثال، يمكن عرض رسالة توضح تطور إجراءات المعالجة فى التطبيقات التى تتطلب وقتا فى التنفيذ.

ونظرا لأن متحكم Label لا يحوز بؤرة التركيز، لذا يمكن استخدامه لتكوين مفاتيح وصول إلى أدوات التحكم الأخرى. وتستخدم خاصية Text لعرض النص الذى يحتوى علية متحكم Label، كما تتيح لنا خاصية Alignment صف النص داخل المتحكم.

تكوين مفاتيح الوصول باستخدام متحكم Label

يمكن استخدام متحكم Label لتعريف مفاتيح وصول خاصة بأدوات التحكم الأخرى. ويترتب على الضغط على مفتاح الوصول مع مفتاح ALT، نقل بؤرة التركيز إلى المتحكم الذى يلى متحكم Label فى ترتيب الجدولة. ويرجع السبب فى ذلك إلى أن متحكم Label لا يستطيع حيازة بؤرة التركيز، ولذلك ينتقل التركيز تلقائيا إلى المتحكم الذى يليه فى ترتيب الجدولة. ويمكن استخدام هذه التقنية لتخصيص مفاتيح وصول إلى مربعات النصوص، مربعات السرد المركبة، مربعات السرد، وشبكات البيانات.

لتخصيص مفتاح وصول إلى متحكم باستخدام متحكم Label:

١. نسحب متحكم Label أولا، ثم نسحب متحكم أخر. أو نضع أدوات التحكم بأي ترتيب ثم نضبط خاصية TabIndex فى متحكم Label لتكون أقل بمقدار واحد

عن المتحكم الأخر.

١. نضبط خاصية خاصية UseMnemonic على True فى متحكم Label.

٢. نستخدم رمز (&) قبل الحرف الذى يمثل مفتاح الوصول فى خاصية Text بمتحكم Label.

```
Label1.UseMnemonic = True
Label1.Text = "&Print"
Label2.UseMnemonic = True
Label2.Text = "&Copy && Paste"
```

تغيير حجم متحكم Label ليتناسب مع محتوياته

يمكن أن يحتوى متحكم Label على سطر أو عدة سطور، ويمكن أن يكون ثابت الحجم أو يمكن أن يغير حجمة ليتناسب مع عنوانه. استخدام خاصية AutoSize تمكن هذا المتحكم من تغيير حجمة تلقائيا.

لجعل متحكم Label يغير حجمة تلقائيا لكى يتناسب مع محتوياته:

١. نختار متحكم Label على النموذج.

٢. نضبط خاصية AutoSize فى نافذة Properties على القيمة True. وفى حالة ضبط قيمة هذه الخاصية على القيمة False، لا يتمدد المتحكم ليتناسب مع النص الذى يحتوى علية ولكن النص قد يستمر على السطر الثانى إذا كان ذلك ممكنا.

مميز الربط (LinkLabel)

يسمح لنا متحكم LinkLabel بإضافة روابط مماثلة لروابط الوب إلى تطبيقات الويندوز. ويمكن استخدام هذا المتحكم لكل شئ يمكن استخدام متحكم Label معه، ويمكن أيضا استخدام جزء من النص للربط مع كائن أو صفحة من صفحات الوب على شبكة الإنترنت. وبالإضافة إلى كل الخصائص، الوسائل، والأحداث الخاصة بمتحكم Label، يحتوى متحكم LinkLabel أيضا على خصائص خاصة بالربط مع الكائنات الأخرى. خاصية LinkArea، تحدد المنطقة التى بها النص الذى ينفذ عملية الربط. خاصية VisitedLinkColor، خاصية LinkColor، وخاصية ActiveLinkColor تضبط ألوان الربط. ويقع حدث LinkClicked عند النقر على نص الربط. وابسط استخدام لمتحكم LinkLabel هو عرض ربط منفرد باستخدام

خاصية LinkArea. ويمكن أيضا تكوين روابط متعددة باستخدام خاصية Links. ويمكن استخدام خاصية LinkData لتحديد بيانات خاصة بكل كائن Link على انفراد. وتستخدم هذه الخاصية للاحتفاظ بموقع ملف لعرضة أو عنوان موقع وب.

تغيير شكل متحكم LinkLabel

يمكن تغيير النص الذى يعرضة متحكم LinkLabel ليتناسب مع أغرض متعددة. على سبيل المثال، من المعتاد الإشارة إلى أن النص يمكن النقر علية بضبطه لكى يبدو بلون محدد تحته خط. وبعد النقر علية، يتغير اللون إلى لون مختلف. للتحكم فى هذا السلوك، يمكن ضبط خمسة خصائص مختلفة، LinkArea, LinkBehavior, LinkColor, VisitedLinkColor, LinkVisited.

لتغيير شكل متحكم LinkLabel:

نضبط خاصية LinkColor وخاصية VisitedLinkColor على الألوان المرغوب بها. يمكن القيام بذلك باستخدام نافذة Properties أثناء التصميم، كما يمكن تنفيذه باستخدام الكود التالى:

- نضبط اللون باستخدام قيم عشرية للأحمر، الأخضر، والأزرق.

```
LinkLabel1.LinkColor = Color.FromArgb (0, 0, 255)
```

كما يمكن ضبط اللون باستخدام ثوابت الألوان المعرفة فى النظام.

```
LinkLabel1.VisitedLinkColor = Color.Purple
```

- نضبط خاصية Text لكى تحتوى على النص المناسب.

```
LinkLabel1.Text = "Click here to see more."
```

- نضبط خاصية LinkArea لتحديد جزء العنوان الذى سوف يستخدم فى الربط. ويتم تمثيل ذلك باستخدام كائن من تصنيف Point يحتوى على رقمين. الرقم الأول يمثل موقع رمز البداية والرقم الثانى يمثل عدد الحروف فى عنوان الاتصال.

```
Dim myPoint As Point
myPoint = New Point (6, 4)
LinkLabel1.LinkArea = myPoint
```

- نضبط خاصية LinkBehavior على قيمة AlwaysUnderline، قيمةHoverUnderline أو NeverUnderline. يترتب على استخدام قيمة HoverUnderline، وضع خط

أسفل ذلك الجزء من العنوان الذى تحدده خاصية LinkArea عند وضع مؤشر الماوس فوقة.

- فى إجراء معالجة حدث LinkClicked، نضبط خاصية LinkVisited على القيمة True.

```
Protected Sub LinkLabel1_LinkClicked (ByVal sender As Object, _
   ByVal e As EventArgs) Handles LinkLabel1.LinkClicked
   LinkLabel1.LinkVisited = True
End Sub
```

الربط مع كائن أو صفحة وب باستخدام متحكم LinkLabel

يسمح لنا متحكم LinkLabel بتكوين روابط على النموذج مثل روابط الوب. وعند النقر على أحد هذه الروابط، يمكن تغيير لونها للإشارة إلى أن الرابطة قد تم زيارتها.

للربط مع نموذج أخر باستخدام متحكم LinkLabel:

١. نضبط خاصية Text لكى تحتوى على عنوان محدد.

٢. نضبط خاصية LinkArea لتحديد الجزء المستخدم فى الربط من العنوان.

٣. فى معالج حدث LinkClicked، نستدعى وسيلة Show لفتح نموذج أخر فى المشروع، ونضبط خاصية LinkVisited على القيمة True.

```
Protected Sub LinkLabel1_LinkClicked (ByVal Sender As System.Object, _
   ByVal e As System.EventArgs) Handles LinkLabel1.LinkClicked
   Dim f2 As New Form ()
   f2.Show
   LinkLabel1.LinkVisited = True
End Sub
```

ويمكن أيضا استخدام متحكم LinkLabel لعرض صفحة وب من خلال المتصفح الإفتراضى. للقيام بذلك، ننفذ الخطوات التالية:

١. نضبط خاصية Text على عنوان محدد.

٢. نضبط خاصية LinkArea فى متحكم LinkLabel على جزء العنوان المستخدم فى الربط.

٣. فـي إجـراء معالجـة حـدث LinkClicked، نضـبط خاصـية LinkVisited على القيمة True ونسـتخدم وسـيلة Process.Start لـبدء المتصـفح الإفتراضى مع إدخال عنوان URL. ولكـي نسـتطيع استخدام وسيلة Process.Start، نحتاج إلى إضافة مرجع إلى منطقة أسماء System. Diagnostics.

```
Protected Sub LinkLabel1_LinkClicked (ByVal Sender As Object, _
  ByVal e As EventArgs) Handles LinkLabel1.LinkClicked
  LinkLabel1.LinkVisited = True
  System.Diagnostics.Process.Start ("http://www.Microsoft.com")
End Sub
```

مربع السرد (ListBox)

يقوم متحكم ListBox بعرض قائمة من البنود التى يستطيع المستخدم اختيار واحد أو أكثر من بينها. فإذا كان مجموع البنود يزيد على العدد الممكن عرضة، يتم تلقائيا إضافة شريط تدرج إلى المتحكم المذكور. عند ضبط خاصية MultiColumn على القيمة True، يعرض مربع السرد البنود فى عدد من الأعمدة مع إضافة شريط تدرج أفقى. وعندما تكون قيمة هذه الخاصية False، يتم عرض البنود فى عمود واحد وإضافة شريط تدرج رأسى. وعندما تكون قيمة خاصية ScrollAlwaysVisible تساوى True، يظهر شريط التدرج بغض النظر عن عدد البنود. وتحدد خاصية SelectionMode عدد البنود التى يمكن اختيارها فى المرة الواحدة.

وتعيد لنا خاصية SelectedIndex قيمة صحيحة تمثل البند الأول الذى تم اختياره فى مربع السرد. ويمكن باستخدام الكود تغيير البند الذى تم اختياره عن طريق تغيير قيمة خاصية SelectedIndex. وسوف يتم تركيز الضوء على البند المقابل فى مربع السرد. وعند عدم اختيار أحد البنود، سوف تحتوى خاصية SelectedIndex على القيمة (1-). وعند اختيار البند الأول فى القائمة، تحتوى خاصية SelectedIndex على القيمة صفر. وعند اختيار أكثر من بند فى القائمة، تحتوى خاصية SelectedItems على أول بند تم اختياره فى مجموعة البنود المختارة. وتماثل خاصية SelectedItem خاصية SelectedIndex، ولكنها تعيد نص البند فى صورة سلسلة من الحروف. تعكس قيمة خاصية Items.Count عدد البنود فى القائمة. وتكون هذه القيمة دائما أكثر بواحد من أكبر قيمة فى فهرس القائمة لأن الفهرس يبدأ بالصفر. ولإضافة أو حذف بنود فى متحكم ListBox، نستخدم وسائل

Items.Clear, Items.Insert, Items.Add, Items.Remove. ويمكن فى وقت التصميم استخدام خاصية Items لإضافة بنود جديدة.

قائمة سرد الايقونات (ListView)

تعرض قائمة ListView قائمة تحتوى على بنود وأيقونات خاصة بها. يمكننا استخدام هذا المتحكم لتكوين واجهة استخدام مثل الجانب الأيمن فى مستكشف الويندوز. ويحتوى هذا المتحكم على أربعة أنماط: أيقونة كبيرة (LargIcon)، أيقونة صغيرة (SmallIcon)، قائمة سرد (List)، وقائمة تفصيلات (Details). يعرض نمط LargeIcon أيقونات كبيرة بجانب نصوص البنود. وتظهر البنود فى أعمدة متعددة عندما يكون المتحكم ضخما بما فيه الكفاية. ويماثل نمط SmallIcon النمط السابق، فيما عدا أنة يعرض أيقونات صغيرة. ويعرض نمط List أيقونات صغيرة ولكنة يستخدم عمودا واحدا فقط. ويعرض نمط Details البنود فى أعمدة متعددة. ويتحدد نمط العرض عن طريق خاصية View. ويمكن لجميع الأنماط أن تعرض صورا من قائمة صور (ImageList).

والخاصية الرئيسية فى متحكم ListView هى خاصية Items، التى تحتوى على البنود التى يعرضها المتحكم. وتحتوى خاصية SelectedItems على مجموعة البنود التى يتم اختيارها فى المتحكم. ويمكن للمستخدم أن يختار عدة بنود عند ضبط خاصية MultiSelect على القيمة True. ويمكن لمتحكم ListView أن يعرض مربعات اختيار بجانب البنود، عن طريق ضبط خاصية CheckBoxes على القيمة True.

وتحدد خاصية Activation نوع الإجراء الذى يجب على المستخدم تنفيذه لتنشيط أحد البنود فى القائمة. والخيارات المتاحة فى هذا السياق هى: TwoClick, OneClick, Standard . يتطلب خيار OneClick النقر مرة واحدة بالماوس لتنشيط البند. ويتطلب خيار TwoClick النقر المزدوج على البند. فى هذه الحالة، تقوم النقرة الأولى بتغيير لون نص البند. ويتطلب خيار Standard النقر المزدوج لتنشيط البند، إلا أن شكل نص البند لا يتغير.

إضافة وحذف البنود بمتحكم ListView

تتكون عملية إضافة بند إلى متحكم ListView بصفة أساسية من تحديد البند ثم تحديد خصائصه. ويمكن استخدام نافذة Properties فى وقت التصميم لإضافة وحذف البنود. كما

يمكن استخدام الكود للقيام بنفس المهمة.

لإضافة أو حذف أحد البنود فى وقت التصميم:

١. فى نافذة Properties، ننقر على نقاط الحذف (...) التالية لخاصية Items. يترتب على ذلك فتح نافذة ListViewItem Collection Editor.

٢. لإضافة أحد البنود، ننقر على زر Add. ويمكن ضبط خصائص البند بعد إضافته، مثل خاصية Text وخاصية ImageIndex.

٣. لحذف بند من القائمة، نختار البند ثم ننقر على زر Remove.

لإضافة بند جديد باستخدام الكود:

```
ListView1.Items.Add ("List item text", 3)
```

لحذف البنود باستخدام الكود، نستخدم وسيلة RemoveAt لحذف أحد البنود، ووسيلة Clear لحذف جميع البنود فى قائمة:

```
ListView1.Items.RemoveAt (0)
ListView1.Items.Clear ()
```

إضافة أعمدة إلى متحكم ListView

يمكن أن يعرض متحكم ListView عدة أعمدة لكل بند فى القائمة عندما يكون فى نمط التفصيلات (Details View). ويمكن استخدام الأعمدة لعرض أنواع عديدة من المعلومات خاصة بكل بند فى القائمة. على سبيل المثال، يمكن أن تعرض قائمة الملفات اسم الملف، نوع الملف، الحجم، وأخر تاريخ تعديل للملف.

لإضافة أعمدة إلى القائمة فى وقت التصميم:

١. نختار متحكم ListView على سطح النموذج.

٢. نضبط خاصية View على القيمة Details.

٣. فى نافذة Properties، ننقر على نقاط الحذف التالية لخاصية Columns. يؤدى ذلك إلى ظهور نافذة ColumnHeader Collection Editor.

٤. نستخدم زر Add لإضافة أعمدة جديدة. ويمكن فى هذه الحالة، اختيار رأس العمود وضبط النص الخاص به، تصفيف النص، والعرض.

لإضافة أعمدة باستخدام الكود:

١. نضبط خاصية View على القيمة Details.

```
ListView1.View = View.Details
```

٢. نستخدم وسيلة Add في خاصية Columns. الكود التالى يضيف عمود عرضة ٢٠.

```
ListView1.Columns.Add ("File type", 20, HorizontalAlignment.Left)
```

عرض أيقونات فى متحكم ListView

يمكن أن يعرض ListView أيقونات من ثلاثة قوائم من الصور. مشاهد List, Details, SmallIcon تعرض صور من قائمة الصور التى تحددها خاصية SmallImageList. ويعرض مشهد LargeIcon صورا من قائمة الصور التى تحددها خاصية LargeImageList. ويمكن أن يعرض مشهد List مجموعة أخرى من الأيقونات التى تحددها خاصية StateImageList بجانب الأيقونات الصغيرة أو الكبيرة.

لعرض صور في مشهد قائمة:

١. نضبط الخاصية المناسبة، LargeImageList، SmallImageList، أو StateImageList على قائمة الصور الموجودة التى نرغب فى استخدامها. ويمكن ضبط هذه الخصائص فى وقت التصميم أو باستخدام كود مماثل للكود التالى:

```
ListView1.SmallImageList = ImageList1
```

٢. نضبط خاصية ImageIndex أو خاصية StateImageIndex لكل بند فى القائمة يرتبط مع أيقونة. ويمكن ضبط هذه الخصائص فى وقت التصميم داخل نافذة ListViewItem Collection Editor، التى يمكن الوصول إليها بالنقر على نقاط الحذف التالية لخاصية Items فى نافذة Properties. ويمكن أيضا ضبط هذه الخصائص باستخدام الكود.

```
ListView1.Items (0).ImageIndex = 3
```

عرض بنود فرعية فى أعمدة باستخدام متحكم ListView

يمكن أن يعرض متحكم ListView نص إضافي، أو بنود فرعية لكل بند فى مشهد التفصيلات (Details View). يعرض العمود الأول نص البند الأساسى. العمود الثانى

والأعمدة التى تليه تعرض البند الفرعى الأول، الثانى، والبنود الفرعية المرتبطة التالية.

لإضافة بنود فرعية إلى بند فى قائمة ListView:

١. نضيف الأعمدة التى نحتاج إليها. ونظرا لأن العمود الأول سوف يعرض نص البند، نحتاج إلى عمود أخر بالإضافة إلى أعمدة البنود الفرعية.

٢. نستدعى وسيلة SetSubItem فى خاصية Items. يقوم الكود التالى بتحديد بندين فرعيين هما: اسم الموظف والقسم التابع له.

```
ListView1.Items (0).SubItems.Add ("John Smith")
ListView1.Items (0).SubItems.Add ("Accounting")
```

تقويم الشهر (MonthCalendar)

يقدم هذا المتحكم واجهة رسومية للمستخدم لمشاهدة وضبط معلومات التاريخ. ويعرض تقويما على شكل شبكة تحتوى على أرقام أيام الشهر مرتبة تحت أيام الأسبوع، كما يقوم بتركيز الضوء على الأيام التى يتم اختيارها. ويمكن اختيار شهر مختلف بالنقر على أزرار الأسهم على جانبى العنوان. وعلى عكس متحكم DateTimePicker، يمكن اختيار أكثر من تاريخ باستخدام هذا المتحكم.

ويقوم هذا المتحكم بعرض تاريخ اليوم فى دائرة، كما يتم كتابته فى أسفل الشبكة. ويمكننا تغيير هذه السمة عن طريق ضبط خاصية ShowToday وخاصية ShowTodayCircle على القيمة False. ويمكن أيضا إضافة أرقام الأسابيع إلى التقويم عن طريق ضبط خاصية ShowWeekNumbers على القيمة True. وعن طريق ضبط خاصية CalendarDimensions، يمكن عرض عدد من الشهور أفقيا ورأسيا. ويظهر يوم الأحد تلقائيا على أنة اليوم الأول فى الأسبوع، ولكن يمكن تغيير ذلك بضبط خاصية FirstDayOfWeek. ويمكن تحديد بعض التواريخ لكى تظهر فى بنط أسود عريض على أساس تاريخ واحد، شهريا أو سنويا، بإضافة كائنات DateTime إلى خاصية AnnuallyBoldDates، خاصية BoldDates، و MonthlyBoldDates.

والخاصية الرئيسية فى متحكم MonthCalendar هى خاصية SelectionRange، التى تحتوى على نطاق التواريخ التى تم اختيارها فى المتحكم. و لايمكن أن تتعدى قيمة هذه

الخاصية الحد الأقصى العدد التواريخ التى يمكن اختيارها، والتى يجرى تحديدها باستخدام خاصية MaxSelectionCount. وأول وأخر تاريخ يمكن للمستخدم اختيارها، تتقرر بضبط خصائص Maxdate و MinDate.

تغيير شكل عرض متحكم MonthCalendar

يمكن تغيير الشكل المعروض به متحكم MonthCalendar عن طريق تغيير اللون، عرض أو إخفاء أرقام الأسبوع والتاريخ الجاري.

لتغيير ألوان التقويم الشهرى، نضبط الخصائص التالية:

- خاصية TitleForeColor التى تحدد لون حروف الأيام والأسبوع.
- خاصية TitleBackColor التى تحدد لون خلفية الأيام والأسبوع.
- خاصية TrailingForeColor التى تحدد لون التواريخ التى تسبق و تلى الشهر أو الشهور المعروضة.

الكود التالى يوضح ضبط هذه الخصائص باستخدام الكود:

```
MonthCalendar1.TitleBackColor = System.Drawing.Color.Blue
MonthCalendar1.TrailingForeColor = System.Drawing.Color.Red
MonthCalendar1.TitleForeColor = System.Drawing.Color.Yellow
```

لعرض التاريخ الجاري أسفل المتحكم:

نضبط خاصية ShowToday على القيمة True. يوضح الكود التالى كيفية عرض وإخفاء تاريخ اليوم عند النقر المزدوج على النموذج.

```
Private Sub Form1_DoubleClick (ByVal sender As Object, _
ByVal e As System.EventArgs) Handles MyBase.DoubleClick
    MonthCalendar1.ShowToday = Not MonthCalendar1.ShowToday
End Sub
```

ولعرض أرقام الأسبوع:

نضبط خاصية ShowWeekNumbers على القيمة True. ويمكن ضبط هذه الخاصية باستخدام نافذة فى وقت التصميم باستخدام نافذة Properties، أو باستخدام الكود. ويترتب على ضبط هذه الخاصية، ظهور أرقام الأسابيع فى عمود منفصل على يسار اليوم الأول فى الأسبوع.

```
MonthCalendar1.ShowWeekNumbers = True
```

إظهار أكثر من شهر فى متحكم MonthCalendar

يمكن أن يعرض متحكم MonthCalendar حتى اثني عشر شهرا فى المرة الواحدة. فى الوضع الإفتراضى، يعرض هذا المتحكم شهرا واحدا فقط، ولكن يمكن تحديد عدد الشهور التى يتم عرضها وكيفية ترتيبها داخل المتحكم. ويتغير حجم المتحكم عندما نقوم بتغيير أبعاد التقويم. لعرض عدد من الشهور، نضبط خاصية CalendarDimensions على عدد الشهور المطلوب عرضها أفقيا أو رأسيا، كما يتضح من الكود التالى:

```
MonthCalendar1.CalendarDimensions = New System.Drawing.Size (3,2)
```

عرض بعض الأيام بالبنط الأسود العريض فى متحكم MonthCalendar

يمكن عرض أيام محددة فى متحكم MonthCalendar بالبنط الأسود العريض. وقد نحتاج إلى ذلك لجذب انتباه المستخدمين إلى تواريخ خاصة، مثل أيام الإجازات وأيام نهايات الأسبوع. ويمكن تنفيذ ذلك باستخدام ثلاثة خصائص:

- خاصية BoldDates، التى تحتوى تواريخ منفردة.
- خاصية AnnuallyBoldDates، التى تحتوى على التواريخ التى تظهر فى بنط أسود عريض كل سنة.
- خاصية MonthlyBoldDates، التى تحتوى على التواريخ التى تظهر فى بنط أسود عريض كل شهر.

وتحتوى كل من هذه الخصائص على مصفوفة من كائنات DateTime. ولجعل أحد التواريخ يظهر فى بنط أسود عريض، نكون كائنات من نوع DateTime، كما يتضح من الكود التالى:

```
Dim myVacation1 As Date = New DateTime (2001, 6, 10)
Dim myVacation2 As Date = New DateTime (2001, 6, 17)
```

لعرض تاريخ منفرد فى بنط أسود عريض، نستدعى إحدى الوسائل التالية:

- وسيلة AddBoldedDate لعرض تاريخ فى بنط أسود عريض.
- وسيلة AddAnnuallyBoldedDate لعرض تاريخ فى بنط أسود عريض كل سنة.
- وسيلة AddMonthlyBoldedDate، لعرض تاريخ فى بنط أسود عريض كل شهر.

الكود التالى، يوضح إضافة تواريخ بالبنط الأسود العريض.

```
MonthCalendar1.AddBoldedDate (myVacation1)
MonthCalendar1.AddBoldedDate (myVacation2)
```

ويمكن عرض مجموعة من التواريخ بالبنط الأسود العريض عن طريق تكوين مصفوفة من كائنات DateTime وتخصيصها لأحد الخصائص، كما يتضح من الكود التالى.

```
Dim VacationDates As DateTime () = {myVacation1, myVacation2}
MonthCalendar1.BoldedDates = VacationDates
```

ولجعل أحد التواريخ يظهر بالبنط العادي، نستخدم إحدى الوسائل الثلاثة التالية المقابلة للوسائل السابقة:

- وسيلة RemoveBoldedDate.
- وسيلة RemoveAnnuallyBoldedDate.
- وسيلة RemoveMonthlyBoldedDate.

يبين الكود التالى مثالا على حذف البنط الأسود العريض.

```
MonthCalendar1.RemoveBoldedDate (myVacation1)
MonthCalendar1.RemoveBoldedDate (myVacation2)
```

ويمكن تغيير جميع التواريخ المعروضة بالبنط الأسود العريض باستخدام واحدة من الخصائص التالية:

- خاصية RemoveAllBoldedDates.
- خاصية RemoveAllAnnuallyBoldedDates.
- خاصية RemoveAllMonthlyBoldedDates.

اختيار نطاق تواريخ فى متحكم MonthCalendar

فى متحكم MonthCalendar يمكن للمستخدم اختيار نطاق من التواريخ باتباع الخطوات التالية:

١. نكون كائنات من تصنيف DateTime تمثل التاريخ الأول والتاريخ الأخير فى النطاق، كما يتضح من الكود التالى:

```
Dim projectStart As Date = New DateTime (2001, 2, 13)
Dim projectEnd As Date = New DateTime (2001, 2, 28)
```

٢. نضبط خاصية SelectionRange، كما يوضحه الكود التالى:

```
MonthCalendar1.SelectionRange = New SelectionRange (projectStart, projectEnd)
```

٣. أو نضبط خاصية SelectionStart وخاصية SelectionEnd، كما يتضح من الكود التالى:

```
MonthCalendar1.SelectionStart = projectStart
MonthCalendar1.SelectionEnd = projectEnd
```

مدرج النطاق الرقمى (NumericUpDown)

يتكون متحكم NumericUpDown من مربع نص وزوجين من الأسهم، يمكن للمستخدم النقر عليهما لتعديل قيمة. ويقوم هذا المتحكم بعرض وضبط قيمة رقمية منفردة فى قائمة من الخيارات. ويمكن للمستخدم أن يزيد وينقص الرقم عن طريق النقر على زر Up وزر Down، الضغط على مفتاح UP ومفتاح DOWN، أو بطباعة رقم فى مربع النص. وأهم الخصائص بهذا المتحكم هى خاصية Value، خاصية Maximum، خاصية Minimum، وخاصية Increment. القيمة الافتراضية لخاصية Maximum تساوى ١٠٠، والقيمة الافتراضية لخاصية Minimum تساوى صفر، والقيمة الافتراضية لخاصية Increment تساوى واحد. وتقوم خاصية Value بتحديد الرقم الذى يتم اختياره فى المتحكم. كما أن خاصية Increment تحتفظ بقيمة التغير الذى يحدث عند النقر على أزرار Up و Down. والوسائل الرئيسية فى هذا المتحكم هى وسيلة UpButton ووسيلة DownButton.

ضبط وإعادة القيم الرقمية باستخدام متحكم NumericUpDown

تحدد خاصية Value القيمة الرقمية فى متحكم NumericUpDown. ويمكننا كتابة اختبارات شرطية لهذة القيمة مثل أى خاصية أخرى. وبمجرد ضبط قيمة هذه الخاصية، يمكن تعديلها مباشرة باستخدام الكود، أو يمكن استدعاء وسيلة UpButton ووسيلة DownButton لتعديل قيمة هذه الخاصية.

لضبط القيمة الرقمية، نخصص قيمة لخاصية Value باستخدام نافذة Properties، أو باستخدام الكود التالى:

```
NumericUpDown1.Value = 55
```

كما يمكن ضبط قيمة خاصية Value باستدعاء الكود التالى:

```
NumericUpDown1.UpButton ()
```

وللحصول على قيمة خاصية Value، يمكننا استخدام كود يماثل الكود التالى:

```
If NumericUpDown1.Value >= 65 Then
  MessageBox.Show ("Age is: " & NumericUpDown1.Value.ToString)
Else
  MessageBox.Show ("The customer is ineligible for a senior citizen discount.")
End If
```

ضبط صيغة متحكم NumericUpDown

يمكن تهيئة كيفية عرض القيم فى متحكم NumericUpDown باستخدام خاصية DecimalPlaces، وخاصية ThousandsSeparator. تحدد خاصية DecimalPlaces عدد الأرقام التى تظهر على يسار العلامة العشرية. والقيمة الافتراضية هى الصفر. وتحدد خاصية ThousandsSeparator إدراج فاصلة كل ثلاثة أرقام عشرية. والقيمة الافتراضية هى False.

لتحديد صيغة القيمة الرقمية، نضبط خاصية DecimalPlaces على رقم صحيح. ونضبط خاصية ThousandsSeparator على القيمة True أو القيمة False.

```
NumericUpDown1.DecimalPlaces = 2
NumericUpDown1.ThousandsSeparator = True
```

اللوحة (Panel)

يستخدم متحكم Panel لتوفير مجموعات من أدوات التحكم الأخرى. ويستخدم هذا النوع من أدوات التحكم فى العادة، لتقسيم النموذج على أساس وظيفى. وعند تحريك متحكم Panel فى وقت التصميم، تتحرك كل أدوات التحكم التى يحتوى عليها أيضا. ويمكن الوصل إلى أدوات التحكم الموضوعة على متحكم Panel باستخدام خاصية Controls. ويماثل متحكم Panel متحكم GroupBox، غير أن متحكم Panel يمكن أن يحتوى على شرائط تدرج (Scroll Bars)، بينما يحتوى متحكم GroupBox على عنوان. ولإظهار شرائط التدرج بمتحكم Panel، نضبط خاصية AutoScroll على القيمة True. ويمكن التحكم فى شكل ظهور متحكم Panel بضبط خاصية BackColor، خاصية BackGroundImage، وخاصية BorderStyle. وتتحكم خاصية BorderStyle فى شكل حدود اللوحة، غير مرئية (None)، خط عادى (FixedSingle)، أو خط مظلل (Fixed3D).

تكوين مجموعة أدوات باستخدام متحكم Panel

هناك ثلاثة أسباب لوضع أدوات التحكم فى مجموعات. السبب الأول هو وضع الأدوات ذات العلاقة فى مجموعة واحدة لجعل واجهة الاستخدام أكثر وضوحا، والسبب الثانى هو التجميع لأغراض البرمجة، والسبب الأخير هو تحريك أدوات التحكم فى وحدة واحدة وقت التصميم.

لتكوين مجموعة من أدوات التحكم باستخدام متحكم Panel:

١. نسحب متحكم Panel من مربع ToolBox إلى النموذج.

٢. نضيف أدوات تحكم أخرى إلى متحكم Panel عن طريق سحب كل متحكم إلى داخل متحكم Panel. وعندما تكون هناك أدوات تحكم موجودة على سطح النموذج ونريد وضعها فى متحكم Panel ، يمكن قص هذه الأدوات ولصقها داخل متحكم Panel، كما يمكن أيضا سحبها إلى داخلة.

٣. وعندما نريد إضافة حدود لمتحكم Panel، نضبط خاصية BorderStyle باختيار واحدة من ثلاثة خيارات: None, FixedSingle, Fixed3D.

ضبط خلفية المتحكم Panel

يمكن أن يعرض متحكم Panel لون خلفية وصورة بهذه الخلفية. تقوم خاصية Color بضبط لون الخلفية بالنسبة لأدوات التحكم التى يحتوى عليها، مثل أدوات Labels، أدوات Radio Buttons. وعند عدم ضبط خاصية BackGroundImage، سوف تمتلئ الخلفية باللون الذى تحدده الخاصية السابقة. وعند ضبط خاصيةBackGroundImage، سوف يتم عرض الصورة خلف أدوات التحكم الموجودة باللوحة.

لضبط خلفية متحكم Panel باستخدام مصمم النماذج:

١. نختار متحكم Panel على سطح النموذج.

٢. فى نافذة Properties، ننقر زر السهم التالى لخاصية BackColor لعرض نافذة بها ثلاثة ملصقات.

٣. نختار ملصق Custom لعرض لوحة ألوان، ونختار ملصق Web أو System لعرض

قائمة بالأسماء المميزة لهذة الألوان.

٤. نختار أحد الألوان.

٥. فى نافذة Properties، ننقر السهم التالى لخاصية BackGroundImage لعرض مربع حوار Open.

٦. نختار ملف الرسومات التى نريد عرضة.

ويمكن ضبط الخلفية باستخدام الكود باستخدام خاصية BackColor لتحديد لون الخلفية:

```
Panel1.BackColor = Color.AliceBlue
```

ونستخدم خاصية BackGroundImage لتحديد الرسم المستخدم. الكود التالى يوضح ذلك، مع مراعاة تغيير المسار الخاص بملف الرسومات:

```
Dim path As String = " C:\program files\messenger\lvback.gif"
Panel1.BackgroundImage = Image.FromFile (path)
```

مربع الصورة (PictureBox)

يستخدم هذا المتحكم لعرض الرسومات فى صيغة metafile ، صيغة JPEG، صيغة GIF صيغة bitmap، أو صيغة Icon. وتحدد خاصية Image الصورة التى يتم عرضها. ويمكن ضبط هذه الخاصية وقت التشغيل أو فى وقت التصميم. وتحكم خاصية SizeMode كيفية توافق الصورة والمركب معا.

تحميل الصورة وقت التصميم

باستخدام متحكم PictureBox ، يمكننا تحميل وعرض صورة على نموذج فى وقت التصميم عن طريق ضبط خاصية Image لكى تحتوى على صورة صالحة.

لعرض صورة وقت التصميم:

١. نسحب متحكم PictureBox إلى النموذج.

٢. فى نافذة Properties، نختار خاصية Image ثم ننقر على نقاط الحذف لعرض مربع حوار Open.

٣. نختار نوع الملف المطلوب فى مربع Files of Type.

٤. نختار اسم الملف الذى نريد عرضة.

لحذف الصورة فى وقت التصميم:

١. فى نافذة Properties، نختار خاصية Image وننقر بزر الماوس الأيمن على الرسم الصغير الذى يظهر على يسار اسم كائن الصورة ثم نختار Reset.

تغيير حجم وموقع الصورة وقت التشغيل

عند استخدام متحكم PictureBox على نموذج، يمكننا ضبط خاصية SizeMode لتحتوى على إحدى القيم التالية:

- صف ركن الصورة الأعلى على اليسار مع نفس الركن فى المتحكم.
- وضع الصورة فى مركز المتحكم.
- تعديل حجم المتحكم ليتلاءم مع حجم الصورة التى يعرضها.
- بسط الصورة لكى تناسب المتحكم.

ويمكن أن ينتج عن بسط الصورة انخفاض فى نوعية الصورة، خصوصا عند استخدام صيغة Bitmap. وتعتبر صيغة Metafiles التى تتكون من مجموعة من الأوامر الخاصية برسم الصور أكثر ملاءمة لبسط الصور.

لضبط خاصية SizeMode فى وقت التشغيل:

نضبط خاصية SizeMode على قيمة CenterImage، قيمة AutoSize، قيمة Normal، أو قيمة StretchImage. تعنى القيمة Normal وضع الصورة فى الركن الأعلى على يسار المتحكم. وتعنى القيمة CenterImage وضع الصورة فى مركز المتحكم. وتعنى قيمة AutoSize تعديل حجم المتحكم تلقائيا ليتناسب مع حجم الصورة. وتعنى قيمة StretchImage تغيير حجم الصورة ليتناسب مع حجم المتحكم.

```
Private Sub StretchPic ()
    Dim path As String = "C:\Windows\Gone Fishing.bmp"
    PictureBox1.SizeMode = PictureBoxSizeMode.StretchImage
    PictureBox1.Image = Image.FromFile (path)
End Sub
```

ضبط الصورة وقت التشغيل

يمكن ضبط الصورة المعروضة بواسطة متحكم PictureBox عن طريق ضبط خاصية

Image باستخدام وسيلة FromFile فى تصنيف Image. الكود التالى يوضح ذلك، مع مراعاة تعديل مسار الملف إلى المسار الصحيح:

```
Private Sub LoadNewPict ()
  Dim path As String = " C:\Windows\  GreenStone.bmp"
  PictureBox1.Image = Image.FromFile (path)
End Sub
```

ولحذف صورة من متحكم PictureBox، نضبط خاصية Image بدون تحديد اسم الملف.

```
PictureBox1.Image = Nothing
```

مربع حوار الطباعة المبدئية (PrintPreviewDialog)

يستخدم هذا المتحكم فى عرض كيفية ظهور إحدى الوثائق عند طباعتها. ويحتوى هذا المتحكم على أزرار للطباعة ، للتكبير والتصغير، لعرض صفحة أو عدة صفحات، وإغلاق مربع الحوار. والخاصية الأساسية فى هذا المكون هى خاصية Document، التى تحدد الوثيقة التى يتم طباعتها مبدئيا. ويجب أن يكون المستند أو الوثيقة من كائنات تصنيف PrintDocument. ولكى يتم عرض مربع الحوار، يجب استدعاء وسيلة ShowDiaolg.

ويحتوى متحكم PrintPreviewDialog على متحكم PrintPreviewControl، الذى يحتوى على عدد من الخصائص المتاحة. من أهم هذه الخصائص، خاصية Columns وخاصية Rows اللتان تحددان عدد الصفحات المعروضة أفقيا ورأسيا. ويمكن الوصول إلى هذه الخصائص باستخدام كود يماثل الكود التالى:

```
PrintPreviewDialog1.PrintPreviewControl.Columns
```

متحكم الطباعة المبدئية (PrintPreviewControl)

يستخدم متحكم PrintPreviewControl لعرض كائن PrintDocument بنفس الصورة التى سوف يظهر بها أثناء الطباعة الفعلية. ولا يحتوى هذا المتحكم على أزرار أو العناصر الأخرى للتعامل مع المستخدم. على هذا الأساس، لا نقوم باستخدام هذا المتحكم إلا فى حالة الرغبة فى كتابة كود خاص بنا يحدد واجهة الطباعة المبدئية الخاصة بنا. واستخدام واجهة التعامل النظامية الخاصة بالطباعة المبدئية، نستخدم متحكم PrintPreviewDialog. وأهم خاصية فى هذا المتحكم هى خاصية Document، التى تضبط مستند الطباعة المبدئية. ويجب أن يكون المستند كائن من تصنيف PrintDocument. كما تحدد خاصية Columns

وخاصية Rows عدد الصفحات المعروضة أفقيا ورأسيا فى هذا المتحكم.

متحكم مؤشر التنفيذ (ProgressBar Control)

يشير متحكم ProgressBar فى نماذج الويندوز إلى مستوى التنفيذ لأحد البرامج أو التطبيقات عن طريق عرض عدد مناسب من المستطيلات المرتبة فى شريط أفقي. وعندما يكتمل البرنامج، يصبح الشريط معبأ بالكامل. ويستخدم هذا المتحكم عموما لإعطاء فكرة للمستخدم عن الوقت الذى يجب انتظاره قبل انتهاء العملية. من أمثلة ذلك، استخدام هذا المتحكم عند تحميل أحد الملفات الكبيرة.

وأهم الخصائص التى يحتوى عليها هذا المتحكم: خاصية Minimum، خاصية Value، وخاصية Maximum. تقوم خاصية Minimum بضبط القيمة الدنيا، بينما تقوم خاصية Maximum بضبط القيمة القصوى التى يمكن أن يعرضها هذا المتحكم. وفى هذه الحالة، يتم تمثيل القيمة الصغرى بمستطيل واحد. ولتحديث قيمة النشاط الجاري، يجب كتابة كود لضبط خاصية Value. على سبيل المثال، عند تحميل ملف ضخم فى ذاكرة الكمبيوتر، يمكن تحديد القيمة القصوى بما يساوى حجم الملف بالكيلو بايت. فإذا تم ضبط القيمة القصوى على 100، وضبط القيمة الصغرى على 10، وضبط خاصية Value على القيمة 50، فإن القيمة المعروضة تساوى خمسة مستطيلات، وهو ما يساوى نصف الكمية الممكن عرضها.

يستخدم المثال التالى متحكم ProgressBar للتحكم فى عرض تطور عملية نسخ مجموعة من الملفات. ويستخدم المثال خاصية Minimum وخاصية Maximum لتحديد مدى خاص بذلك المتحكم، يساوى عدد الملفات التى يجرى نسخها. ويستخدم المثال أيضا، خاصية Step مع وسيلة PerformStep لزيادة خاصية Value بعد نجاح نسخ ملف. ويفترض المثال وجود متحكم ProgressBar يسمى pBar1 وأن هناك وسيلة تسمى CopyFile تقوم بعملية النسخ وتعيد قيمة منطقية تدل على نجاح نسخ الملف. ويفترض الكود أيضا أن هناك مصفوفة تحتوى على الملفات المطلوب نسخها. كما يفترض تمرير هذه المصفوفة إلى وسيلة CopyWithProgress وأن هذه الوسيلة يتم استدعائها من وسيلة أخرى أو إجراء معالجة حدث فى النموذج.

```
Private Sub CopyWithProgress (ByVal ParamArray filenames As String ())
عرض متحكم ProgressBar.
  pBar1.Visible = True
ضبط خاصية Minimum على القيمة واحد لتمثيل الملف الأول الذى يتم نسخة.
  pBar1.Minimum = 1
ضبط خاصية Maximum على قيمة تساوى عدد الملفات التى سوف يتم نسخها.
  pBar1.Maximum = filenames.Length
ضبط القيمة الابتدائية فى متحكم ProgressBar.
  pBar1.Value = 1
ضبط خاصية Step على القيمة واحد.
  pBar1.Step = 1
استطلاع جميع الملفات المطلوب نسخها.
  Dim x As Integer
  for x = 1 To filenames.Length – 1
نسخ الملف وزيادة قيمة المؤشر عند نجاح النسخ.
    If CopyFile(filenames(x - 1)) = True Then
      pBar1.PerformStep()
    End If
  Next x
End Sub
```

زر الراديو (RadioButton)

تقدم أدوات التحكم من نوع RadioButton مجموعة من اثنين أو أكثر من الخيارات المتبادلة المنفردة أمام المستخدم. وقد تبدو أدوات التحكم من نوع CheckBox ونوع RadioButton متشابهة الوظيفة، إلا أن هناك فرقا مهما بينهما يتمثل فى عدم القدرة على اختيار أكثر من متحكم RadioButton فى نفس الوقت بنفس المجموعة، على عكس أدوات التحكم من نوع CheckBox. ويعنى تعريف مجموعة من أزرار الراديو، توجيه رسالة إلى المستخدم بأن هناك مجموعة من الخيارات التى يمكنة اختيار واحد منها فقط.

وعند النقر على متحكم RadioButton، يتم ضبط خاصية Checked على القيمة True، كما يتم استدعاء إجراء معالجة حدث Click. ويقع حدث CheckedChanged عند تغيير

قيمة خاصية Checked. وإذا تم ضبط خاصية AutoCheck على القيمة True عند اختيار RadioButton، فإن كل اختيارات أزرار الراديو الأخرى فى المجموعة يتم إزالتها تلقائيا. ويتم التحكم فى النص المعروض داخل متحكم RadioButton باستخدام خاصية Text، التى يمكن أن تحتوى على اختصارات مفاتيح وصول. ويمكن ضبط خاصية Appearance على القيمة Button لكى يعرض متحكم RadioButton بعرض رسومات باستخدام خاصية Image وخاصية ImageList.

تكوين مجموعات من متحكم RadioButton

لقد تم تصميم أدوات التحكم من نوع RadioButton لكى توفر للمستخدم خيارا واحدا من بين عدد من الخيارات. ويجرى تكوين مجموعات أزرار الراديو عن طريق سحبها إلى داخل حاوية، مثل متحكم Panel، متحكم GroupBox، أو متحكم Form. كل أزرار الراديو التى يتم إضافتها مباشرة إلى نموذج، تصبح مجموعة واحدة. ولتكوين مجموعات منفصلة من هذه الأزرار، نحتاج إلى وضعها على متحكم Panel أو متحكم GroupBox.

لوضع أزرار الراديو فى مجموعة منفصلة:

١. نسحب متحكم GroupBox أو متحكم Panel من مربع ToolBox إلى النموذج.

٢. نضع أدوات RadioButton على متحكم Panel أو متحكم GroupBox.

المثال التالى يستنسخ ويعد متحكم RadioButton، يضبط خاصية AutoCheck على القيمة False ويضيف المتحكم إلى نموذج.

```
Private Sub InitializeMyRadioButton ()
   Dim radioButton1 As New RadioButton ()
   radioButton1.Appearance = Appearance.Button
   radioButton1.AutoCheck = False
   Controls.Add (radioButton1)
End Sub
```

مربع النص الزكى (RichTextBox)

يستخدم هذا المتحكم فى عرض، إدخال، ومعالجة النصوص ذات الصيغ. ويقوم المتحكم بكل ما يقوم به متحكم مربع النص (TextBox)، بالإضافة إلى إمكانية عرض أطقم الحروف، الألوان، الوصلات، تحميل النص والصور التى يحتوى عليها ملف. وفى

الأساس، يستخدم هذا المتحكم فى معالجة النصوص وعرض سمات مشابهة لتطبيقات معالجة الكلمات، مثل تطبيق Microsoft Word. ويمكن أن يعرض متحكم RichTextBox شرائط التدرج. والخاصية التى تتحكم فى النص المعروض هى خاصية Text. بجانب هذه الخاصية، يحتوى هذا المتحكم على عدد أخر كبير من الخصائص التى تستخدم فى صياغة النص. وتقوم وسيلة LoadFile ووسيلة SaveFile بعرض وكتابة صيغ ملفات عديدة. ويمكن استخدام وسيلة Find للعثور على سلسلة من النصوص أو حروف محددة. ويمكن أيضا استخدام متحكم RichTextBox فى الربط المماثل لروابط الوب عن طريق ضبط خاصية DetectUrls على القيمة True، وكتابة كود لمعالجة حدث LinkClicked. ويمكن منع المستخدم من معالجة بعض أو كل النصوص فى متحكم RichTextBox عن طريق ضبط خاصية SelectionProtected على القيمة True. كما يمكن استخدام خاصية Undo وخاصية Redo للتعامل مع حذف النصوص وإلغاء الحذف السابق. وتمكننا وسيلة CanRedo من تحديد ما إذا كان فى الإمكان استخدام العملية التى سبق حذفها.

القيمة	الإيضاح
Both	لعرض شرائط Scroll bars رأسيا وأفقيا
None	لا يتم عرض شرائط Scroll bars
Horizontal	عرض شريط Scroll bar الأفقى
Vertical	عرض شريط Scroll bar رأسيا
ForcedHorizontal	عرض شريط Scroll bar الأفقى عندما تساوى خاصية WordWrap القيمة False
ForcedVertical	عرض شريط Scroll bar الرأسى دائما
ForceBoth	عرض شريط Scroll Bar الرأسى دائما، والأفقي عندما تساوى خاصية WordWrap القيمة False

جدول ١٩

شرائط التدرج فى متحكم RichTextBox

فى الوضع الإفتراضى يقوم متحكم RichTextBox بعرض شرائط التدرج الأفقية والرأسية عند الحاجة. ولعرض شرائط التدرج، نتبع الخطوات التالية:

١. نختار متحكم RichTextBox فى مصمم النماذج.

٢. فى نافذة Properties، نضبط خاصية MultiLine على القيمة True.

٣. نضبط خاصية ScrollBars على قيمة مناسبة فى تعداد RichTextBoxScrollBars التى يعرضها الجدول رقم (١٩).

٤. نضبط خاصية WordWrap على قيمة True لتوفيق النص مع المتحكم، أو على القيمة False.

عرض الروابط المماثلة لروابط الوب

يمكن أن يعرض متحكم RichTextBox روابط وب بإظهارها فى لون مختلف ووضع خط تحتها. ويمكن كتابة الكود اللازم لفتح نافذة مستعرض (Internet Explorer) لعرض موقع الوب المحدد فى نص الارتباط عند النقر علية. لربط صفحة وب مع متحكم RichTextBox، ننفذ الإجراءات التالية:

١. نضبط خاصية Text على سلسلة نص تحتوى على عنوان URL.

٢. نضبط خاصية DetectUrls على القيمة True.

٣. نكتب إجراء معالجة لحدث LinkClicked، كما يوضحه الكود التالى:

```
Private Sub RichTextBox1_LinkClicked (ByVal sender As Object, ByVal e As System.Windows.Forms.LinkClickedEventArgs) Handles RichTextBox1.LinkClicked
    System.Diagnostics.Process.Start (e.LinkText)
End Sub
```

عمليات السحب والإسقاط فى متحكم RichTextBox

تحدث عمليات السحب والإسقاط باستخدام متحكم RichTextBox عن طريق معالجة حدث DragEnter. وتعتبر هذه العمليات بسيطة مع متحكم RichTextBox، ولا توجد حاجة إلى كتابة كود لمعالجة حدث DragDrop. الخطوات اللازمة لتمكين عمليات السحب والإسقاط فى متحكم RichTextBox تشمل:

١. نضبط خاصية AllowDrop في المتحكم على القيمة True.

٢. نكتب كود في إجراء معالجة حدث DragEnter. ونستخدم عبارة If للتحقق من صلاحية نوع البيانات التي يتم سحبها. كما يمكن ضبط خاصية DragEventArgs.Effect على أي قيمة في تعداد DragDropEffects، كما يتضح من الكود التالي:

```
Private Sub RichTextBox1_DragEnter (ByVal sender As Object, ByVal e As System.Windows.Forms.DragEventArgs) Handles RichTextBox1.DragEnter
  If (e.Data.GetDataPresent (DataFormats.Text)) Then
    e.Effect = DragDropEffects.Copy
  Else
    e.Effect = DragDropEffects.None
  End If
End Sub
```

تحميل الملفات في متحكم RichTextBox

يمكن استخدام وسيلة LoadFile لتحميل البيانات من تيار بيانات (Stream)، باتباع الخطوات التالية:

١. نحدد مسار الملف الذي سوف نقوم بفتحة. يمكننا استخدام مكون OpenFileDialog للقيام بهذه المهمة.

٢. نستدعي وسيلة LoadFile، مع تحديد اسم الملف. ويمكن أيضا تحديد نوع الملف. وعندما يتم استدعاء هذه الوسيلة بدون تحديد نوع الملف، يجري استخدام النوع الإفتراضي وهو RTF. ولتحديد نوع ملف أخر، نستدعي الوسيلة المذكورة مع استخدام قيمة من تعداد RichTextBoxStreamType في المعامل الثاني من الوسيلة.

```
RichTextBox1.LoadFile ("C:\MyDocs\Testdoc.txt", _
  RichTextBoxStreamType.PlainText)
```

حفظ الملفات باستخدام متحكم RichTextBox

يمكن أن يقوم متحكم RichTextBox بحفظ المعلومات التي يعرضها باستخدام وسيلة SaveFile. لتنفيذ حفظ محتويات متحكم RichTextBox في ملف، نتبع الخطوات التالية:

١. نحدد مسار الملف الذي نريد استخدامه في الحفظ.

٢. نستدعى وسيلة SaveFile مع تحديد اسم الملف الذى يتم حفظة وتحديد نوع هذا الملف اختياريا. وعندما يتم استدعاء هذه الوسيلة وتزويدها باسم الملف فقط، سوف يتم حفظ الملف فى صيغة RTF. ولحفظ الملف باستخدام صيغة أخرى ، نستدعى الوسيلة المذكورة مع إدخال قيمة فى المعامل الثانى من قيم تعداد RichTextBoxStreamType.

```
RichTextBox1.SaveFile ("C:\MyDocs\Testdoc.rtf", _
  RichTextBoxStreamType.RichNoOleObjs)
```

ضبط طاقم الحروف المستخدم فى متحكم RichTextBox

يحتوى متحكم RichTextBox على عدد ضخم من الخيارات المستخدمة فى صياغة النص الذى يتم عرضة. حيث يمكننا جعل الحرف المختار يظهر فى بنط أسود عريض، تحته خط، أو مائلا باستخدام خاصية SelectionFont. ويمكن أيضا استخدام هذه الخاصية لتغيير حجم ونوع الحروف المختارة. وتسمح لنا خاصية SelectionColor بتغيير ألوان الحروف المختارة.

لتغيير شكل عرض الحروف:

١. نضبط خاصية SelectionFont على طاقم الحروف المناسب.

٢. نضبط خاصية SelectionColor على لون مناسب.

```
RichTextBox1.SelectionFont = New Font("Tahoma", 12, FontStyle.Bold)
RichTextBox1.SelectionColor = System.Drawing.Color.Red
```

ويجب ملاحظة أن هذه الخصائص تؤثر فقط على الحروف التى يجرى اختيارها، وإذا لم يتم اختيار نص فإنها تؤثر على النص الذى يتم طباعته فى موقع نقطة الإدراج.

شريط التدرج (ScrollBar)

يستخدم متحكم ScrollBar فى التطبيقات أو أدوات التحكم، لتوفير التنقل السهل داخل قائمة طويلة من البنود أو كمية كبيرة من المعلومات عن طريق التدرج فى العرض أفقيا ورأسيا. وتعتبر شرائط التدرج من الأدوات الشائعة الاستخدام عند تكوين واجهات التعامل مع المستخدمين (Interfaces). وهناك نوعان من هذا المتحكم: متحكم HScrollBar، ومتحكم VScrollBar. ويعمل كل من متحكم HscrollBar ومتحكم VScrollBar بطريقة منفصلة

عن أدوات التحكم الأخرى ولدى كل منهما مجموعة أحداث وخصائص ووسائل خاصة. ويجب الإنتباة إلى أن هذه الأدوات ليست نفس شرائط التدرج المستخدمة داخليا مع أدوات التحكم الأخرى، مثل متحكم ComboBox، متحكم ListBox، ومتحكم TextBox. ويستخدم متحكم ScrollBar حدث Scroll لمراقبة حركة مربع التدرج (Scroll Box)، الذى يطلق علية الإبهام (The Thumb)، على طول شريط التدرج. واستخدام حدث Scroll يتيح الوصول إلى قيم شريط التدرج عند الحركة علية.

خاصية Value

خاصية Value هى قيمة صحيحة تمثل موقع المربع على شريط التدرج، وقيمتها الافتراضية تساوى صفر. وعندما يكون المربع فى أقصى اليسار على شريط التدرج الأفقى أو فى أعلى الشريط بالنسبة لشريط التدرج الرأسى، تحتوى خاصية Value على أدنى قيمة. وعندما تحتوى خاصية Value على أقصى قيمة، يكون مربع التدرج فى أقصى اليمين على شريط التدرج الافقى وفى أسفل الشريط بالنسبة لشريط التدرج الرأسى. وتعنى القيمة المتوسطة فى خاصية Value، وجود مربع التدرج فى منتصف الشريط.

ويمكن تغيير القيمة على شرط التدرج بالنقر الماوس، كما يمكن سحب مربع التدرج إلى أى نقطة على الشريط. وتعتمد القيمة الناتجة على موقع مربع التدرج، ولكنها تكون دائما فى نطاق قيمة خاصية Minimum و قيمة خاصية Maximum التى يحددها المستخدم.

خاصية LargeChange وخاصية SmallChange

عندما يقوم المستخدم بالضغط على مفتاح PAGEUP أو مفتاح PAGEDOWN أو النقر داخل مسار شريط التدرج على جانبى المربع، تتغير قيمة خاصية Value على أساس القيمة المحددة فى خاصية LargeChange. وعندما يضغط المستخدم على أحد مفاتيح الأسهم أو ينقر على أحد أزرار شريط التدرج، تتغير قيمة خاصية Value على أساس قيمة خاصية SmallChange.

متحكم المقسم (Splitter Control)

يستخدم متحكم Splitter لتغيير حجم أدوات التحكم المستقرة (Docked) على أحد الجوانب فى نموذج، أثناء تشغيل التطبيق. ويستخدم هذا المتحكم فى الغالب على النماذج

التى تحتوى على أدوات تحكم تعرض بيانات متغيرة الحجم، مثل مستكشف الويندوز (Windows Explorer)، الذى تحتوى أقسامه على بيانات متغيرة الحجم فى أوقات متغيرة. وعندما يضع المستخدم مؤشر الماوس على الجانب غير المستقر للمتحكم الذى يمكن تغيير حجمة بواسطة متحكم Splitter، يتغير شكل المؤشر ليشير إلى أن المتحكم يمكن تغيير حجمة. ويسمح متحكم Splitter للمستخدم بتغيير حجم المتحكم الذى يستقر أمامه مباشرة فى نموذج ويندوز. ولهذا يجب استقرار المتحكم على أحد الجوانب فى النموذج واستقرار متحكم Splitter على الجانب غير المستقر من النموذج لكى نمكن المستخدم من تغير حجمة. لتطبيق ذلك عمليا، نتبع الخطوات التالية:

١. نكون تطبيق ويندوز جديد.
٢. نسحب متحكم TreeView من مربع ToolBox إلى النموذج الإفتراضى.
٣. باستخدام نافذة Properties، نضبط خاصية Dock على اليسار بالنقر على الجانب الأيسر فى محرر القيمة، الذى يظهر عند النقر على السهم المتجه إلى أسفل.
٤. نسحب متحكم Splitter من مربع الأدوات إلى النموذج. سوف يستقر Splitter تلقائيا على الحافة اليمنى من متحكم TreeView.
٥. نسحب متحكم Panel من مربع الأدوات إلى النموذج. وفى نافذة Properties، نضبط خاصية Dock على القيمة Fill بالنقر على القسم الأوسط فى محرر القيمة، الذى يظهر عند النقر على السهم المتجه إلى أسفل. يترتب على ذلك شغل متحكم Panel لكامل الجانب الأيمن من النموذج.
٦. نسحب متحكم ListView من مربع الأدوات إلى متحكم Panel السابق. نجعل خاصية Dock فى متحكم ListView تساوى Top.
٧. نسحب متحكم Splitter من مربع الأدوات إلى متحكم Panel. فى نافذة Properties ، نضبط خاصية Dock على القيمة Top بالنقر على القسم الأعلى فى محرر القيمة. يؤدى ذلك إلى استقرار متحكم Splitter أسفل متحكم ListView.
٨. نسحب متحكم RichTextBox من مربع الأدوات إلى متحكم Panel. نضبط خاصية

Dock على القيمة Fill ثم نضغط مفتاح F5 لتشغيل التطبيق.

شريط المعلومات (StatusBar)

يستخدم متحكم شريط المعلومات على النموذج باعتباره منطقة تستخدم بواسطة التطبيق لعرض أنواع متعددة من المعلومات. ويمكن أن يحتوى هذا المتحكم على لوحات (Panels) تعرض نصوص أو أيقونات تشير إلى الوضع، أو مجموعة من الأيقونات المتحركة التى تشير إلى أن التطبيق يعمل، مثل الأيقونة التى تشير إلى أن مستند Microsoft Word جارى حفظة. ويستخدم مستكشف الإنترنت هذا الشريط لعرض عنوان الصفحة عند وضع مؤشر الماوس على الوصلة الخاصة بهذه الصفحة.

ويمكننا عرض رسالة واحدة فى شريط المعلومات عن طريق ضبط خاصية ShowPanels على القيمة False وضبط خاصية Text على النص الذى نريد عرضة. ويمكننا تقسيم شريط المعلومات إلى لوحات لعرض أكثر من نوع من المعلومات عن طريق ضبط خاصية ShowPanels على القيمة True واستخدام وسيلة Add فى تصنيف StatusBarPanelCollection لإضافة لوحات إلى الشريط.

إضافة لوحات إلى شريط المعلومات

تتكون المنطقة القابلة للبرمجة فى شريط المعلومات من أمثلة من تصنيف StatusBarPanel، التى يتم إضافتها فى وقت التصميم باستخدام محرر StatusBarPanelCollection. وفى وقت التشغيل، يمكن إضافة أمثلة من التصنيف المذكور من خلال إضافة بنود إلى تصنيف StatusBarPanelCollection.

لإضافة لوحات إلى شريط المعلومات فى وقت التصميم:

١. نضيف متحكم StatusBar إلى النموذج.

٢. فى نافذة Properties، ننقر خاصية Panels لاختيارها ثم ننقر نقاط الحذف على يمين الخاصية لفتح محرر StatusBarPanelCollection.

٣. نستخدم أزرار Add و Remove لإضافة وحذف لوحات فى متحكم StatusBar. ونهيئ خصائص اللوحة المنفردة فى نافذة Properties. وأهم هذه الخصائص هى

AutoSize، Alignment، BorderStyle، Icon، Style، Text، و MinWidth.

٤. ننقر Ok لإقفال مربع الحوار وتكوين اللوحات التى تم تحديدها.

٥. فى نافذة Properties، نضبط خاصية ShowPanels على القيمة True.

لإضافة لوحات إلى شريط المعلومات باستخدام الكود:

نكون لوحات شريط معلومات ونضيفها إلى مجموعة StatusBarPanels فى أحد الإجراءات. نحدد قيم الخصائص لكل لوحة من اللوحات باستخدام فهرس اللوحة الذى يتم تمريرة إلى خاصية Panels. المثال التالى يستخدم الكود لتنفيذ هذه المهام، مع افتراض وجود نموذج به متحكم StatusBar.

```
Public Sub CreateStatusBarPanels ()
  StatusBar1.Panels.Add ("One")
  StatusBar1.Panels.Add ("Two")
  StatusBar1.Panels.Add ("Three")

  StatusBar1.Panels (0).AutoSize = StatusBarPanelAutoSize.Spring
  StatusBar1.Panels (1).AutoSize = StatusBarPanelAutoSize.Contents
  StatusBar1.Panels (2).AutoSize = StatusBarPanelAutoSize.Contents

  StatusBar1.Panels (0).BorderStyle = StatusBarPanelBorderStyle.Raised
  StatusBar1.Panels (1).BorderStyle = StatusBarPanelBorderStyle.Sunken
  StatusBar1.Panels (2).BorderStyle = StatusBarPanelBorderStyle.Raised


  StatusBar1.Panels    (2).Icon    =    New    System.Drawing.Icon("C:\Application
Data\Icon.ico")
  StatusBar1.ShowPanels = True
End Sub
```

تحديد اللوحة التى النقر عليها فى متحكم StatusBar

يمكن برمجة StatusBar لكى يستجيب لنقرات الماوس باستخدام عبارة Select Case داخل معالج حدث PanelClick. يحتوى هذا الحدث على معامل Panel، الذى يحتوى على مرجع إلى كائن StatusBarPanel الذى يتم النقر علية. باستخدام هذا المرجع، يمكننا معرفة فهرس اللوحة التى تم النقر عليها والبرمجة على هذا الأساس. ويجب التحقق من ضبط خاصية ShowPanels على القيمة True.

الكود التالى، يوضح استخدام عبارة Select Case لتحديد اللوحة التى تم النقر عليها فى متحكم StatusBar:

```
Private Sub StatusBar1_PanelClick (ByVal sender As System.Object, ByVal e As System.Windows.Forms.StatusBarPanelClickEventArgs) Handles StatusBar1.PanelClick
  Select Case StatusBar1.Panels.IndexOf (e.StatusBarPanel)
   Case 0
    MessageBox.Show ("You have clicked Panel One.")
   Case 1
    MessageBox.Show ("You have clicked Panel Two.")
  End Select
End Sub
```

ضبط حجم اللوحات فى شريط المعلومات

كل مثل من تصنيف StatusBarPanel داخل متحكم StatusBar يحتوى على عدد من الخصائص الديناميكية التى تحدد حجمة وسلوك إعادة تحديد حجمة فى وقت التشغيل.

لضبط حجم لوحة فى وقت التصميم:

١. فى نافذة Properties، نختار خاصية Panel ثم ننقر على نقاط الحذف لفتح StatusBarPanel Collection Editor.

٢. نضبط الخصائص التى تظهر فى نافذة Properties التى تظهر على الجانب الأيمن فى مربع حوار StatusBarPanel Collection Editor.

لضبط حجم إحدى اللوحات باستخدام الكود:

نضبط خاصية MinWidth، خاصية AutoSize، وخاصية Width المتعلقة باللوحة فى أحد الإجراءات، كما يتضح من الكود التالى:

```
Public Sub SetStatusBarPanelSize ()
```

تكوين اللوحة

```
  StatusBar1.Panels.Add ("One")
```

ضبط الخصائص

```
  StatusBar1.Panels (0).AutoSize = StatusBarPanelAutoSize.Spring
  StatusBar1.Panels (0).Width = 200
```

عرض اللوحات

```
  StatusBar1.ShowPanels = True
End Sub
```

الملصق (TabControl)

يستخدم هذا المتحكم لعرض عدد من الصفحات المميزة بملصقات. ويمكن أن تحتوى كل صفحة على صور وعلى أدوات تحكم أخرى. ويمكن استخدام متحكم TabControl لتكوين مربع حوار متعدد الصفحات، مثل الذى يظهر فى أماكن عديدة بنظام تشغيل الويندوز. والخاصية الأكثر أهمية فى متحكم TabControl هى TabPages، التى تحتوى على الصفحات المنفردة. وكل صفحة هى كائن من نوع TabPage. وعند النقر على أحد الملصقات يقع حدث Click الخاص بكائن الصفحة.

إضافة متحكم إلى الصفحة الخاصة بأحد الملصقات

يستخدم متحكم TabControl لعرض أدوات تحكم أخرى بطريقة منظمة. ويمكن إضافة أدوات تحكم إلى صفحة ملصق فى وقت التصميم، باتباع الخطوات التالية:

١. ننقر الملصق المناسب لوضع الصفحة الخاصة به أمام الصفحات الأخرى.

٢. نضع متحكم على الصفحة باستخدام مربع الأدوات.

ولإضافة متحكم إلى صفحة ملصق باستخدام الكود، نستخدم وسيلة Add فى خاصية Controls بكائن TabPage كما يتضح من الكود التالى:

```
Dim MyButton as New Button ()
TabControl1.TabPages (0).Controls.Add (MyButton)
```

إضافة وحذف الملصقات باستخدام TabControl

يمكن إضافة وحذف ملصقات باستخدام مصمم النماذج أو باستخدام الكود. لإضافة ملصق فى وقت التصميم:

١. نسحب متحكم TabControl من مربع الأدوات إلى النموذج.

٢. فى نافذة Properties، ننقر على Add Tab أسفل نافذة الخصائص، أو ننقر على نقاط الحذف التالية لخاصية TabPages لكى نفتح نافذة TabPage Collection Editor ثم ننقر زر Add.

لحذف ملصق باستخدام مصمم النماذج:

١. ننقر على الملصق الذى نريد حذفة لوضع الصفحة الخاصة به فوق الصفحات الأخرى.

٢. فى نافذة Properties، ننقر على Remove Tab أسفل نافذة الخصائص. يترتب على ذلك، حذف الملصق المعروض على قمة الملصقات الأخرى. أو ننقر نقاط الحذف التالية لخاصية TabPages لفتح نافذة TabPage Collection Editor. فى الجانب الأيسر من النافذة، تحت Members، نختار الملصق الذى نريد حذفة ثم ننقر Remove.

لإضافة ملصق باستخدام الكود، نستخدم وسيلة Add الموجودة فى خاصية TabPages:

```
Dim myTabPage As New TabPage ()
myTabPage.Text = "TabPage" & (TabControl1.TabPages.Count + 2)
TabControl1.TabPages.Add (myTabPage)
```

لحذف ملصق باستخدام الكود، نستخدم وسيلة Remove التابعة لخاصية TabPages لحذف أحد الملصقات. كما يمكن استخدام وسيلة Clear لحذف جميع الملصقات.

```
TabControl1.TabPages.Remove (TabControl1.SelectedTab)
TabControl1.TabPages.Clear ()
```

تغيير شكل عرض متحكم TabControl

يمكن تغيير شكل عرض الملصقات فى نماذج الويندوز باستخدام كائنات TabControl و TabPage التى تكون الملصقات المنفردة. يشمل ذلك عرض أيقونة فى الملصق، تكوين صفوف متعددة فى الملصقات، ترتيب الملصقات على أحد الجوانب، إخفاء وإظهار الملصقات، وعرض الفواصل فى صورة أزرار.

لعرض أيقونة فى ملصق:

١. نضيف مكون ImageList إلى النموذج.

٢. نضيف رسومات إلى مكون ImageList.

٣. نضبط خاصية ImageList فى متحكم TabControl على اسم مكون ImageList.

٤. نضبط خاصية ImageIndex التابعة لمتحكم TabPage على فهرس الصورة المناسبة.

ولتكوين صفوف متعددة فى الملصقات:

١. نضيف العدد المطلوب من الملصقات.

٢. نضبط خاصية MultiLine التابعة لمتحكم TabControl على القيمة True.

٣. وعندما لا تظهر صفوف متعددة فى الملصقات، نضبط خاصية Width فى متحكم TabControl لكى يكون أضيق من المساحة التى تتطلبها جميع الملصقات.

ولترتيب الفواصل على جانب المتحكم، نضبط خاصية Alignment فى متحكم TabControl على Left أو Right.

لتمكين أو حجب الملصقات باستخدام الكود، نضبط خاصية Enabled فى كائن TabPage على القيمة True أو القيمة False على الترتيب.

```
TabPage1.Enabled = False
```

ويمكن عرض الملصقات فى صورة أزرار عن طريق ضبط خاصية Appearance فى متحكم TabControl على Buttons أو FlatButtons.

مربع النص (TextBox)

تستخدم مربعات النصوص للحصول على الإدخالات من المستخدم أو عرض أحد النصوص. ويستخدم هذا المتحكم بصفة عامة فى تدقيق النصوص. ويوفر متحكم TextBox نمط صياغة واحد للنص المعروض أو الذى يقوم المستخدم بإدخاله فى المتحكم. وعند الرغبة فى إدخال أنماط صياغة متعددة، يمكننا استخدام متحكم RichTextBox.

وتحتوى خاصية Text على النص المعروض فى المتحكم. وعند ضبط خاصية MultiLine على القيمة True، يمكننا عرض صفوف متعددة. ويمكن ضبط خاصية Text فى وقت التصميم باستخدام نافذة Properties. وفى وقت التشغيل يمكن ضبطها باستخدام الكود أو بواسطة إدخالات المستخدم. ويمكن الحصول على محتويات مربع النص فى وقت التشغيل بقراءة خاصية Text. يقوم الكود التالى بضبط خاصية Text فى وقت التشغيل:

```
Private Sub InitializeMyControl ()
  TextBox1.Text = "This is a TextBox control."
```

```
End Sub
```

التحكم فى نقطة الادراج فى متحكم TextBox

عندما يحوز متحكم TextBox التركيز، فى البداية سوف تقع نقطة الإدراج الافتراضية فى مربع النص على يسار أى نص موجود. ويمكن للمستخدم تحريك هذه النقطة باستخدام لوحة المفاتيح أو الماوس. وإذا فقد مربع النص التركيز ثم حصل علية مرة أخرى، سوف تكون نقطة الإدراج فى الموقع السابق الذى وضعها المستخدم به قبل انتقال البؤرة بعيدا عنها. وفى بعض الحالات قد لا يتلاءم هذا السلوك مع المستخدم. فى هذه الحالة، يمكن استخدام خاصية SelectionStart وخاصية SelectionLength فى تعديل هذا السلوك لكى يتناسب مع احتياجاتنا.

للتحكم فى نقطة الإدراج، نضبط خاصية SelectionStart على قيمة مناسبة. استخدام القيمة صفر يؤدى إلى وضع نقطة الإدراج قبل أول حرف فى النص. ويمكن ضبط خاصية SelectionLength اختياريا على قيمة تمثل طول النص الذى نريد اختياره. يضبط الكود التالى نقطة الإدراج دائما الصفر.

```
Private Sub TextBox1_Enter (ByVal sender As Object, ByVal e As System.EventArgs) Handles TextBox1.Enter
  TextBox1.SelectionStart = 0
  TextBox1.SelectionLength = 0
End Sub
```

تكوين مربع كلمة مرور باستخدام متحكم TextBox

مربع كلمة المرور هو مربع نص يعرض مواقع محجوزة للحروف التى يقوم المستخدم بإدخالها. ولتكوين مربع كلمة المرور، نتبع الخطوات التالية:

١. نضف متحكم TextBox إلى النموذج ونختاره.

٢. فى نافذة Properties، نضبط خاصية PasswordChar على أحد الحروف الذى يعرض فى مربع النص بدلا من الحروف التى يدخلها المستخدم.

٣. نضبط خاصية MaxLength إختياريا. تحدد هذه الخاصية عدد الحروف التى يمكن طباعتها فى المربع.

يقوم الكود التالى بإعداد مربع نص لقبول سلسلة من ١٤ حرف ويعرض بدلا منها رمز

النجمة. ولن يتم تنفيذ هذا الكود إلا إذا تم استدعاء إجراء InitializeMyControl الذى يحتوى علية.

```
Private Sub InitializeMyControl ()
  TextBox1.Text = ""
  TextBox1.PasswordChar = "*"c
  TextBox1.MaxLength = 14
End Sub
```

تكوين مربع نص للقراءة فقط

يمكن تحويل مربع نص يقبل تدقيق النصوص إلى مربع لقراءتها فقط عن طريق ضبط خاصية ReadOnly فى متحكم TextBox على القيمة True. يترتب على ذلك توفر إمكانية قراءة النص ونسخة، ولكن لايمكن تغيير النص. كما يمكن نسخ محتويات TextBox فى هذه الحالة، ولايمكن اللصق به أو القطع منة.

استخدام علامات التنصيص فى سلاسل النصوص

يمكن وضع علامة التنصيص فى سلسلة نص فى الكود الذى نقوم بكتابته، باتباع إحدى الطرق التالية. الطريقة الأولى تشمل إدراج علامتين من علامات التنصيص فى نص، كما يتضح من الكود التالى:

```
Private Sub InsertQuote ()
  TextBox1.Text = "She said, ""You deserve a treat!"" "
End Sub
```

الطريقة الثانية تشمل إدراج علامة التنصيص فى صيغة ASCII أو صيغة UNICODE ، كما يوضحه الكود التالى:

```
Private Sub InsertAscii ()
  TextBox1.Text = "She said, " & Chr (34) & "You deserve a treat!" & Chr (34)
End Sub
```

الطريقة الثالثة تشمل تعريف ثابت يمثل الحرف واستخدام هذا الثابت فى مكان الحرف عند الحاجة:

```
Const quote As String = """"
TextBox1.Text = "She said, " & quote & "You deserve a treat!" & quote
```

إختيار نص فى متحكم TextBox باستخدام الكود

يمكن إختيار نص فى متحكم TextBox باستخدام الكود. على سبيل المثال، يمكن

إختيار سلسلة النص التى نبحث عنها عند العثور عليها فى أحد النصوص، لتنبيه المستخدم إلى موقع السلسلة فى ذلك النص. ولاختيار نص باستخدام الكود:

١. نضبط خاصية SelectionStart على بداية السلسلة التى نريد اختيارها. وتمثل قيمة هذه الخاصية رقم يشير إلى نقطة الإدراج داخل النص.

٢. نضبط خاصية SelectionLength على قيمة طول النص الذى نريد إختيارة. جعل هذه القيمة أكبر من الصفر يؤدى إلى اختيار عدد الحروف الذى تحددة هذه القيمة ، بدءا من الموقع الحالى لنقطة الإدراج.

٣. يمكن الوصول إلى النص المختار باستخدام خاصية SelectedText.

الكود التالى يقوم باختيار محتويات مربع نص عند وقوع حدث Enter فى متحكم TextBox:

```
Private Sub TextBox1_Enter (ByVal sender As Object, ByVal e As System.EventArgs) Handles TextBox1.Enter
  TextBox1.SelectionStart = 0
  TextBox1.SelectionLength = TextBox1.Text.Length
End Sub
```

عرض صفوف متعددة فى متحكم TextBox

فى الوضع الإفتراضى، يعرض متحكم TextBox سطرا واحدا فقط من النصوص ولايعرض أشرطة Scroll Bars. فإذا كان النص أكبر من المسافة المتاحة ، يمكننا مشاهدة بعض النص فقط. ويمكن تغيير هذا السلوك الإفتراضى عن طريق ضبط خاصية WordWrap، خاصية MultiLine ، خاصية ScrollBars على القيم المناسبة. لمشاهدة سطور متعددة:

١. نضبط خاصية MultiLine على القيمة True. فإذا كانت خاصية WordWrap تساوى True ، فإن النص الموجود بالمتحكم سوف يظهر في صيغة فقرة واحدة أو أكثر. وإذا كانت خاصية WordWrap تساوى False ، تظهر النصوص فى مربع النص فى صورة قائمة سرد مقصوصة السطور من الجانب.

٢. نضبط خاصية ScrollBars على قيمة None، قيمة Horizontal ، أو قيمة Both.

٣. نضبط خاصية WordWrap على قيمة True أو قيمة False.

شريط الأدوات (ToolBar)

يستخدم متحكم ToolBar على نماذج الويندوز ليكون شريط تحكم يعرض صف من القوائم المنسدلة والأزرار ذات الرسومات التى تنفذ الأوامر. وهكذا، فإن النقر على زر فى شريط أدوات يمكن أن يساوى إختيار أمر فى قائمة. ويمكن تهيئة الأزرار لكى تظهر وتعمل فى صورة أزرار أوامر، قوائم منسدلة، أو فواصل. وفى الأساس، يحتوى شريط الأدوات على أزرار وقوائم منسدلة تقابل البنود الموجودة فى هيكل قائمة، مما يوفر طريقة سريعة للوصول إلى أوامر ووظائف التطبيق.

ويجرى دائما تثبيت متحكم ToolBar على طول قمة النافذة الأصلية، ولكن يمكن أيضا تثبيتة على أى جانب بالنافذة. ويمكن تغيير حجم هذا المتحكم وسحبة. ويمكن أيضا استخدام شريط الأدوات فى عرض معلومات مختصرة (ToolTips) عندما يشير المستخدم إلى بمؤشر الماوس على أحد الأزرار فى الشريط. وعند ضبط خاصية Appearance على القيمة Normal، تظهر الأزرار بشريط الأدوات فى صورة بارزة وثلاثية الأبعاد. ويمكن ضبط هذه الخاصية عل القيمة Flat لعرض أزرار الشريط فى صورة مسطحة. وعندما يتحرك مؤشر الماوس فوق أحد الأزرار، يصبح ثلاثى الأبعاد. ويمكن تقسيم الأزرار على شريط الأدوات إلى مجموعات منطقية باستخدام الفواصل. ويمكن تعريف الفاصل بأنة زر شريط أدوات يتم ضبط خاصية Style به على القيمة Separator. ويبدو مثل المسافة الخالية على شريط الأدوات. وعندما يكون شريط الأدوات مسطحا، تظهر الفواصل مثل الخطوط.

ويسمح لنا متحكم ToolBar بتكوين شرائط الأدوات عن طريق إضافة كائنات Button إلى مجموعة Buttons. ويمكن استخدام Collection Editor لإضافة أزرار إلى متحكم ToolBar . ويجب أن يحتوى كل كائن Button على نص أو صورة، ويمكن تخصيص كليهما. ويتم توفير الصور بواسطة مكون ImageList مرتبط مع شريط الأدوات.

وفى وقت التشغيل، يمكن إضافة أو حذف أزرار من مجموعة ToolBarButtonCollection باستخدام وسيلة Add ووسيلة Remove. ولبرمجة هذة الأزرار، نضيف كود إلى إجراء معالجة أحداث ButtonClick التابعة لشريط الأدوات، باستخدام خاصية Button فى تصنيف ToolBarButtonClickEventArgs لتحديد الزر الذى تم النقر

علية.

إضافة أزرار إلى شريط الأدوات

تمثل الأزرار المضافة إلى شريط الأدوات جزءا متكاملا معها. يمكن استخدامها لتوفير وصول سهل إلى أوامر القائمة (The Menu) الموجودة بالتطبيق، كما يمكن استخدام هذه الأزرار لإتاحة أوامر أمام المستخدمين غير متاحة فى القائمة الموجودة بالتطبيق.

لإضافة أزرار فى وقت التصميم:

١. من القائمة المنسدلة فى قمة نافذة Properties، نختار متحكم ToolBar السابق إضافته إلى النموذج.

٢. ننقر خاصية Buttons لاختيارها ثم ننقر نقاط الحذف لفتح نافذة ToolBarButtonCollection Editor.

٣. نستخدم أزرار Add و Remove لإضافة وحذف أزرار فى متحكم ToolBar.

٤. نهيئ خصائص الأزرار المنفردة فى نافذة Properties التى تظهر في الجانب الأيمن من المحرر.

٥. ننقر Ok لإقفال مربع الحوار وتكوين اللوحات التى نقوم بتحديدها.

ولإضافة أزرار باستخدام الكود:

١. نستخدم أحد الإجراءات لتكوين أزرار شريط أدوات وإضافتها إلى مجموعة ToolBarButtons.

٢. نحدد ضوابط خصائص كل زر بتمرير فهرس الزر من خلال خاصية Buttons.

يفترض المثال التالى وجود نموذج به متحكم ToolBar.

```
Public Sub CreateToolBarButtons ()
```

تكوين الأزرار وضبط خاصية Text

```
ToolBar1.Buttons.Add("One")
ToolBar1.Buttons.Add ("Two")
ToolBar1.Buttons.Add ("Three")
ToolBar1.Buttons.Add ("Four")
```

ضبط خصائص لوحات شريط المعلومات

ضبط خاصية Style

```
ToolBar1.Buttons (0).Style = ToolBarButtonStyle.PushButton
ToolBar1.Buttons (1).Style = ToolBarButtonStyle.Separator
ToolBar1.Buttons (2).Style = ToolBarButtonStyle.ToggleButton
ToolBar1.Buttons (3).Style = ToolBarButtonStyle.DropDownButton
```

ضبط خاصية PartialPush

```
ToolBar1.Buttons (2).PartialPush = True
```

استنساخ مكون ContextMenu وMenuItems

```
Dim cm As New ContextMenu ()
Dim miOne As New MenuItem ("One")
Dim miTwo As New MenuItem ("Two")
Dim miThree As New MenuItem ("Three")
cm.MenuItems.Add (miOne)
cm.MenuItems.Add (miTwo)
cm.MenuItems.Add (miThree)
ToolBar1.Buttons (3).DropDownMenu = cm
```

ضبط خاصية Pushed فى الأزرار

```
ToolBar1.Buttons (0).Pushed = True
```

ضبط خاصية ToolTip

```
  ToolBar1.Buttons (1).ToolTipText = "Button 2"
End Sub
```

تخصيص أيقونة لزر فى شريط أدوات

يمكن لأزرار متحكم ToolBar أن تعرض أيقونات داخلها لتسهيل التمييز بواسطة المستخدم. ويتم تحقيق ذلك من خلال إضافة رسومات إلى مكون ImageList ثم ربط هذا المكون مع متحكم ToolBar.

لتخصيص أيقونة لزر بشريط أدوات فى وقت التصميم:

١. نسحب مكون ImageList من مربع ToolBox إلى النموذج.

٢. فى نافذة Properties، ننقر خاصية Images ونضيف رسومات إلى متحكم ImageList.

٣. نسحب متحكم ToolBar من مربع الأدوات إلى النموذج.

٤. فى نافذة Properties، نضبط خاصية ImageList فى متحكم ToolBar على اسم مكون ImageList السابق إضافته.

٥. ننقر بزر الماوس على خاصية Buttons ثم ننقر نقاط الحذف لفتح ToolBarButtonCollection Editor.

٦. نستخدم زر Add لإضافة أزرار إلى متحكم ToolBar.

٧. فى نافذة Properties التى تظهر على يمين نافذة ToolBarButtonCollection Editor، نضبط خاصية ImageIndex لكل زر على شريط الأدوات.

ولضبط أيقونة زر فى شريط الأدوات باستخدام الكود:

١. نستنسخ مكون ImageList ومتحكم ToolBar فى أحد الإجراءات.

٢. فى نفس الإجراء، نخصص أحد الصور لمكون ImageList.

٣. فى نفس الإجراء، نخصص مكون ImageList لمتحكم ToolBar ونضبط خاصية ImageIndex لكل زر على شريط الأدوات.

الكود التالى، ينفذ هذه الخطوات باستخدام الكود:

```
Public Sub InitializeMyToolBar ()
```

استنساخ مكون ImageList

```
   Dim ToolBar1 as New ToolBar
   Dim ImageList1 as New ImageList
```

نخصص صورة لمكون ImageList

```
  Dim myImage As System.Drawing.Image =  Image.FromFile
("C:\winnt\Sample.ico")
   ImageList1.Images.Add (myImage)
```

تكوين زر شريط أدوات

```
   Dim ToolBarButton1 As New ToolBarButton ()
```

نضيف زر شريط الأدوات إلى شريط الأدوات

```
   ToolBar1.Buttons.Add (toolBarButton1)
```

نخصص مكون ImageList لشريط الأدوات

```
  ToolBar1.ImageList = ImageList1
```

نحدد فهرس الصورة

```
  ToolBarButton1.ImageIndex = 0
End Sub
```

إطلاق إحداث القائمة بواسطة أزرار شريط الأدوات

عند استخدام شريط الأدوات، نحتاج إلى معرفة الزر الذى قام المستخدم بالنقر علية. فى حدث ButtonClick بمتحكم ToolBar، يمكننا تقييم خاصية Button فى تصنيف ToolBarButtonClickEventArgs. يوضح الكود التالى خطوات التعامل مع حدث Click على شريط الأدوات:

١. نضيف أزرار إلى متحكم شريط الأدوات

```
Public Sub ToolBarConfig ()
  ToolBar1.Buttons.Add (New ToolBarButton ("One"))
  ToolBar1.Buttons.Add (New ToolBarButton ("Two"))
  ToolBar1.Buttons.Add (New ToolBarButton ("Three"))
```

نضيف إجراء معالجة حدث النقر على زر فى شريط الأدوات

```
  AddHandler ToolBar1.ButtonClick, AddressOf Me.ToolBar1_ButtonClick
End Sub
```

٢. نضيف إجراء معالجة حدث ButtonClick

```
Protected Sub ToolBar1_ButtonClick (ByVal sender As Object, _
ByVal e As ToolBarButtonClickEventArgs)
  Select Case ToolBar1.Buttons.IndexOf (e.Button)
    Case 0
      MessageBox.Show ("First toolbar button clicked")
    Case 1
      MessageBox.Show ("Second toolbar button clicked")
    Case 2
      MessageBox.Show ("Third toolbar button clicked")
  End Select
End Sub
```

مشهد الشجرة (TreeView)

يعرض متحكم TreeView هرم من العقد، مثل الشكل الذى تعرض به الملفات

والمجلدات فى الجانب الأيسر من مستكشف الويندوز. يمكن أن تحتوى كل عقدة على عقد أخرى، تسمى العقد التابعة. والعقد الأصلية أو العقد التى تحتوى على العقد التابعة، يمكن عرضها فى صورة موسعة أو منطوية. ويمكن أيضا عرض شجرة تحتوى على مربعات اختيار بجانب عقد الشجرة، عند ضبط خاصية CheckBoxes على القيمة True. ويمكن فى هذه الحالة استخدام الكود فى اختيار أو إزالة العقد عن طريق ضبط خاصية Checked على القيمة True أو القيمة False. والخصائص الأساسية فى متحكم TreeView هى: خاصية Nodes، وخاصية SelectedNode. خاصية Nodes تحتوى على قائمة بعقد المستوى الأعلى من الشجرة. بينما تحتوى خاصية SelectedNode على العقد المختارة حاليا. ويمكن عرض أيقونات بجانب العقد باستخدام الرسومات الموجودة فى مكون ImageList.

إضافة وحذف العقد فى متحكم TreeView

بالنظر إلى أن متحكم TreeView يعرض عقد فى شكل هرمى، لذا يجب الاهتمام بمعرفة العقدة الأصلية قبل إضافة عقدة تابعة.

لإضافة أو حذف عقد فى وقت التصميم:

١. نختار متحكم TreeView أو نضيف واحدا منها إلى النموذج.

٢. فى نافذة Properties، ننقر على نقاط الحذف التالية لخاصية Nodes. يترتب على ذلك ظهور نافذة TreeNode Editor.

٣. لإضافة عقد، يجب وجود عقدة جذر (Root Node) أولا. وإذا لم يمكن هناك عقدة جذر، يجب إضافة واحدة بالنقر على زر AddRoot. يمكن بعد ذلك إضافة عقد تابعة عن طريق اختيار الجذر أو أى عقدة أخرى ثم النقر على زر Add Child. ولحذف عقدة، نختار العقدة المستهدفة ثم ننقر على زر Delete.

لإضافة عقد باستخدام الكود:

نستخدم وسيلة Add فى خاصية Nodes بمتحكم TreeView، كما يتضح من الكود التالى:

```
Dim newNode As TreeNode = New TreeNode ("Text for new node")
TreeView1.SelectedNode.Nodes.Add (newNode)
```

ولحذف العقد باستخدام الكود:

نستخدم وسيلة Remove فى خاصية Nodes بمتحكم TreeView لحذف عقدة منفردة، أو وسيلة Clear لحذف جميع العقد.

```
TreeView1.Nodes.Remove (TreeView1.SelectedNode)
TreeView1.Nodes.Clear ()
```

تحديد عقدة الشجرة التى تم النقر عليها

عند العمل مع متحكم TreeView، فإن من المهام الشائعة تحديد العقدة التى تم النقر عليها، والاستجابة لذلك. لتحديد العقدة التى النقر عليها فى شجرة:

١. نستخدم كائن EventArgs لإعادة مرجع إلى كائن العقدة التى تم النقر عليها.

٢. نحدد العقدة التى تم النقر عليها عن طريق عرض قيمة خاصية مثل خاصية Text أو خاصية Index.

```
Private Sub TreeView1_AfterSelect (ByVal sender As System.Object, _
ByVal e As System.Windows.Forms.TreeViewEventArgs) Handles
TreeView1.AfterSelect
  MessageBox.Show (e.Node.Text)
End Sub
```

تكرار فحص العقد فى شجرة

من المفيد أحيانا فحص كل عقدة فى متحكم TreeView لكى ننقذ بعض العمليات الحسابية على قيم هذه العقد. يمكن القيام بذلك عن طريق إستخدام إجراء يقوم بتكرار عملية لكل عقدة من العقد فى مجموعة من مجموعات الشجرة. وتحتوى كل عقدة من عقد الشجرة على خصائص يمكن استخدامها فى التجول خلال الشجرة. أهم هذه الخصائص: خاصية FirstNode، خاصية LastNode، خاصية NextNode، خاصية PrevNode، وخاصية Parent. تحدد قيمة خاصية Parent العقدة الأصلية للعقدة الحالية. وتحتوى خاصية Nodes على العقدة الفرعية التابعة إذا كان هناك أى منها. ويحتوى متحكم TreeView ذاتة على خاصية TopNode التى تعتبر عقدة الجذر لكامل الشجرة.

لتنفيذ تكرار الفحص لعقد الشجرة:

١. نكون إجراء يقوم بتكرار اختبار كل عقدة من عقد الشجرة.

٢. نستدعى ذلك الإجراء.

الكود التالى يبين نموذجا لهذا الإجراء الذى يقوم بتكرر فحص عقد الشجرة وطباعة النص الموجود فى خاصية Text بكل كائن من كائنات TreeNode:

```
Private Sub PrintRecursive (ByVal n As TreeNode)
  System.Diagnostics.Debug.WriteLine (n.Text)
  MessageBox.Show (n.Text)
  Dim aNode As TreeNode
  For Each aNode In n.Nodes
    PrintRecursive (aNode)
  Next
End Sub
```

ضبط الأيقونات فى متحكم TreeView

يمكن أن يعرض متحكم TreeView أيقونات على جانب كل عقدة. ويتم وضع الأيقونات مباشرة على يسار نص العقدة. ولعرض هذه الأيقونات، يجب الربط بين متحكم TreeView وبين مكون ImageList.

لعرض أيقونات فى شجرة:

١. نضبط خاصية ImageList فى متحكم TreeView على اسم مكون ImageList الذى نريد استخدامه. ويمكن ضبط هذه الخاصية باستخدام نافذة Properties، كما يمكن ضبطها باستخدام الكود التالى:

```
TreeView1.ImageList = ImageList1
```

٢. نضبط خاصية ImageIndex وخاصية SelectedImageIndex. تحدد خاصية ImageIndex الصورة المرتبطة بالعقدة فى الحالة العادية وفى حالة التوسع. وتحدد خاصية SelectedImageIndex الصورة التى تعرض فى حالة اختيار العقدة.

ويمكن ضبط هذه الخصائص باستخدام الكود، أو فى داخل نافذة TreeNode Editor. لفتح هذه النافذة، ننقر على نقاط الحذف بجانب خاصية Nodes فى نافذة الخصائص. ويمكن ضبط هذه الخصائص باستخدام الكود التالى:

```
TreeView1.SelectedNode.ImageIndex = 0
TreeView1.SelectedNode.SelectedImageIndex = 1
```

أدوات التحكم بدون واجهات التعامل (Components)

تقدم المكونات (Components) أو أدوات التحكم غير المرئية، وظائف مهمة للتطبيقات التى نقوم بتكوينها. وعلى خلاف أدوات التحكم الأخرى، لا تقدم هذه المكونات واجهة تعامل مع المستخدمين. وعلى هذا الأساس، لا تحتاج هذه المكونات إلى العرض على وجه مصمم نماذج الويندوز. وعند إضافة مكون إلى نموذج، يقوم مصمم نماذج الويندوز بعرض مربع متغير الحجم فى أسفل النموذج لعرض جميع المكونات المضافة إلى ذلك النموذج. وبمجرد إضافة المكون إلى هذا المربع، يمكن اختياره وضبط خصائصه باستخدام نافذة Properties أو باستخدام الكود.

مربع حوار الالوان (ColorDialog)

يمثل هذا المكون مربع حوار يسمح للمستخدم باختيار لون من لوحة ألوان وإضافة الألوان التى يعدها إلى هذه اللوحة. وهو نفس مربع حوار الألوان الذى نراه فى تطبيقات الويندوز الأخرى. ويجب استخدام هذا المكون داخل تطبيقاتنا لأنة يمثل طريقة سهلة للتعامل مع الألوان. ونحصل على اللون الذى نقوم باختياره من مربع الألوان فى خاصية Color. ولعرض مربع حوار الألوان، يجب استخدام وسيلة ShowDialog.

تغيير شكل مربع حوار الألوان

يمكن تهيئة الشكل الذى يظهر به مربع حوار الألوان باستخدام عدد من خصائصه. يحتوى هذا المربع على قسمين: القسم الأول يظهر الألوان الأساسية والقسم الأخر يسمح للمستخدم بتكوين ألوان خاصة به. وتقيد معظم الخصائص الألوان التى يستطيع المستخدم اختيارها من مربع الألوان. عند ضبط خاصية AllowFullOpen على القيمة True، فإن المستخدم يستطيع تكوين الألوان الخاصة به. وعند ضبط خاصية AnyColor على القيمة True، يعرض مربع الحوار كل الألوان المتاحة فى مجموعة الألوان الأساسية. وعند ضبط خاصية SolidColorOnly على القيمة True، يستطيع المستخدم اختيار الألوان الصماء فقط. وإذا كانت قيمة خاصية ShowHelp تساوى True، فإن زر Help يظهر على مربع الحوار. وعندما ينقر المستخدم على زر Help، يقع حدث HelpRequest الخاص بمربع حوار الألوان. لتهيئة شكل ظهور مربع حوار الألوان، نضبط قيم الخصائص الموضحة فى الكود

التالى :

```
ColorDialog1.AllowFullOpen = True
ColorDialog1.AnyColor = True
ColorDialog1.SolidColorOnly = False
ColorDialog1.ShowHelp = True
```

يقوم المثال التالى بتكوين مربع حوار ColorDialog. ويفترض هذا المثال وجود نموذج به متحكم TextBox ومتحكم Button.

```
Protected Sub button1_Click (sender As Object, e As System.EventArgs)
   Dim MyDialog As New ColorDialog ()
```

منع المستخدم من تكوين ألوان خاصة به

```
   MyDialog.AllowFullOpen = False
```

تمكين المستخدم من الحصول على المساعدة

```
   MyDialog.ShowHelp = True
```

ضبط اللون الإبتدائى على اللون الحالى للنص

```
   MyDialog.Color = textBox1.ForeColor
   MyDialog.ShowDialog ()
   textBox1.ForeColor = MyDialog.Color
End Sub 'button1_Click
```

مكون القائمة المختصرة (ContextMenu Component)

القائمة المختصر هى مكون يستخدم لإمداد المستخدمين بقائمة سهل الوصول إليها تحتوى على الأوامر التى يتكرر استخدامها، وترتبط هذه القائمة بسياق أو كائن محدد. والبنود التى تحتوى عليها هذه القائمة تكون فى الغالب مجموعة فرعية من بنود قائمة رئيسية تظهر فى أحد أجزاء التطبيق. وتظهر القائمة المختصرة أمام المستخدم عند النقر بزر الماوس الأيمن على الكائن المرتبط بالقائمة. ويمكن إرفاق قائمة مختصر مع أداة تحكم عن طريق ضبط خاصية ContextMenu بالمتحكم على اسم مكون القائمة. ويمكن الربط بين قائمة مختصرة واحدة وبين عدد من أدوات التحكم، ولكن لايمكن الربط بين كائن واحد وبين عدد من القوائم المختصرة.

والخاصية الرئيسية فى مكون القائمة المختصرة هى خاصية MenuItems. ويمكن إضافة بنود إلى القائمة باستخدام محرر القوائم وقت التصميم أو باستخدام الكود. ويتم ذلك عن

طريق تكوين بنود قائمة ثم إضافتها إلى مجموعة MenuItems فى قائمة مختصرة. وبالنظر إلى أن بنود القائمة المختصرة تكون فى الغالب مأخوذة من قوائم أخرى، لذا سوف يتكرر إضافة بنود إلى القائمة المختصرة عن طريق نسخ بنود قوائم أخرى.

وقد تم شرح الموضوعات المتعلقة بالقائمة المختصرة مع الأمثلة التى توضح كيفية التعامل معها فى الفصل الثالث.

كاشف الأخطاء (ErrorProvider Component)

يسمح لنا مكون ErrorProvider بالكشف عن الأخطاء أمام المستخدم. ويستخدم بصفة أساسية مقترنا مع تدقيق إدخالات المستخدم على نموذج الويندوز، أو عرض الأخطاء داخل فئة بيانات. ويعتبر هذا المكون بديلا مفضلا على عرض رسائل الأخطاء فى مربعات الرسائل، لأن استخدام مربع الرسالة يترتب علية فقد الرسالة بمجرد إقفال المربع. ويعرض مكون ErrorProvider أيقونة تدل على الخطأ على جانب المتحكم ذات العلاقة، مثل مربع نص. وعندما يضع المستخدم مؤشر الماوس على هذه الأيقونة، تظهر عبارة مختصرة (ToolTip) تبين سلسلة نص رسالة الخطأ.

والخصائص الأساسية لهذا المكون هى خاصية DataSource، خاصية ContainerControl، وخاصية Icon. ويجب ضبط خاصية ContainerControl على حاوية مناسبة، تكون فى الغالب نموذج الويندوز، لكى نمكن مكون ErrorProvider من عرض أيقونة الخطأ على النموذج. وعند إضافة المكون باستخدام مصمم نماذج الويندوز، يتم ضبط هذه الخاصية على اسم النموذج الذى يحتوى على المكون. وعند إضافة المكون باستخدام الكود، يجب ضبط ذلك بأنفسنا.

ويمكن ضبط خاصية Icon على أيقونة خطأ يقوم المستخدم بإعدادها بدلا من الأيقونة الافتراضية. وعند ضبط خاصية DataSource، يمكن أن يقوم مكون ErrorProvider بعرض رسائل خطأ خاصة بفئات البيانات (Datasets). والوسيلة الأساسية فى مكون ErrorProvider هى وسيلة SetError، التى تحدد سلسلة رسالة الخطأ ومكان ظهور أيقونة الخطأ.

استخدام مكون ErrorProvider لعرض أيقونات أخطأ تدقيق البيانات

يمكن استخدام مكون ErrorProvider لعرض أيقونة خطأ عندما يقوم المستخدم بإدخال بيانات غير صحيحة. ويجب أن يكون لدينا على الأقل اثنان من أدوات التحكم على النموذج لكى نستطيع التنقل بينها باستخدام مفتاح TAB لكى يتم تنفيذ كود التدقيق.

لعرض أيقونة خطأ عندما تكون القيمة فى إحدى أدوات التحكم غير صحيحة:

١. نضيف عدد اثنين من أدوات التحكم ، مربعات نصوص على سبيل المثال، إلى نموذج الويندوز.

٢. نضيف مكون ErrorProvider إلى النموذج.

٣. نختار المتحكم الأول ونضيف كود إلى معالج حدث Validating. ولكى يعمل الكود بطريقة صحيحة، يجب ربط الإجراء بالحدث. الكود التالى يختبر صلاحية البيانات التى يقوم المستخدم بإدخالها. وعندما تكون البيانات غير صحيحة، يتم استدعاء وسيلة SetError. المعامل الأول فى هذه الوسيلة يحدد المتحكم الذى يجب عرض أيقونة الخطأ بجانبه. والمعامل الثانى هو نص الخطأ الذى سوف يتم عرضة.

```
Private Sub TextBox1_Validating (ByVal Sender As Object, ByVal e As CancelEventArgs) Handles TextBox1.Validating
  If Not IsNumeric (TextBox1.Text) Then
    ErrorProvider1.SetError (TextBox1, "Not a numeric value.")
  Else
    ErrorProvider1.SetError (TextBox1, "")
  End If
End Sub
```

٤. نقوم بتشغيل التطبيق. ندخل بيانات غير رقمية فى مربع النص الأول، ثم ننتقل إلى مربع النص الثانى باستخدام مفتاح TAB. عندما يتم عرض أيقونة الخطأ، نشير إليها بمؤشر الماوس لمشاهدة نص الخطأ.

مشاهدة خطأ داخل فئة بيانات باستخدام مكون ErrorProvider

يمكن استخدام مكون ErrorProvider لمشاهدة أخطاء الأعمدة داخل فئة بيانات أو مصدر بيانات أخر. ولكى يستطيع مكون ErrorProvider عرض أخطاء البيانات على

نموذج، ليس من الضروري أن يكون مرتبطا مباشرة مع متحكم على هذا النموذج. بمجرد ربط هذا المكون مع مصدر بيانات، يمكنة عرض أيقونة خطأ بجانب أى متحكم مرتبط مع نفس مصدر البيانات. ويجب الإنتباة إلى أنة عند تغيير مصدر وعضو البيانات فى وقت التشغيل، يجب استخدام وسيلة BindToDataAndErrors لتجنب التناقض.

لعرض أخطاء البيانات:

١. نربط مكون ErrorProvider مع جدول بيانات.

```
TextBox1.Bindings.Add ("Text", DataSet1, "Customers.Name")
ErrorProvider1.DataSource = DataSet1
ErrorProvider1.DataMember = "Customers"
```

٢. نضبط خاصية ContainerControl على اسم النموذج المستخدم

```
ErrorProvider1.ContainerControl = Me
```

٣. نضبط موقع السجل بنموذج الويندوز على صف يحتوى على عمود الخطأ.

```
DataTable1.Rows (5).SetColumnError ("Name", "Bad data in this row.")
BindingManager (DataTable1).Position = 5
```

مربع حوار أطقم الحروف (FontDialog)

مكون FontDialog هو مربع حوار سابق الإعداد يمثل مربع حوار أطقم الحروف القياسى المستخدم فى عرض أطقم الحروف المثبتة فى نظام التشغيل المستخدم. ويستخدم هذا المكون فى تطبيقات الويندوز على أساس أنة الحل الأسهل لاختيار أطقم الحروف بدلا من تكوين مربعات حوار خاصة بنا. يعرض هذا المكون مربعات قوائم لسرد أطقم الحروف، حجمها، مربعات اختيارات خاصة بتأثيرات مثل وضع الخطوط أسفل الحروف، واللغة المستخدمة فى الكتابة. ولعرض مربع حوار FontDialog، نستدعى وسيلة ShowDialog.

وأهم خصائص مربع حوار FontDialog خاصية Font وخاصية Color. تقوم خاصية Font بتحديد طاقم الحروف المستخدم، النمط، الحجم، اللغة، والتأثيرات المستخدمة. على سبيل المثال، طاقم الحروف Arial، الحجم 10pt، النمط Italic، والتأثيرات Strikeout.

يستخدم المثال التالى وسيلة ShowDialog لعرض مربع حوار FontDialog. ويفترض أن هناك نموذجا يوجد علية متحكم TextBox ومتحكم Button.

```
Protected Sub button1_Click (sender As Object, e As System.EventArgs)
```

```
  fontDialog1.ShowColor = True
  If fontDialog1.ShowDialog () <> DialogResult.Cancel Then
     textBox1.Font = fontDialog1.Font
     textBox1.ForeColor = fontDialog1.Color
  End If
End Sub
```

مقدم المساعدة (HelpProvider)

يستخدم مكون HelpProvider لربط ملف مساعدة HTML مع تطبيق ويندوز. ويمكن توفير المساعدة بعدة طرق:

- توفير مساعدة خاصة بأدوات التحكم على نموذج الويندوز.
- توفير مساعدة خاصة بمربع حوار محدد أو أدوات التحكم الموجودة علية.
- توفير مساعدة خاصة بمجالات معينة، مثل الوصول إلى صفحة المحتويات، الفهرس، أو البحث.

إضافة مكون HelpProvider إلى نموذج يسمح لأدوات التحكم الموجودة على النموذج بكشف خصائص المساعدة المتوفرة في مكون HelpProvider. يمكننا ذلك من تقديم المساعدة لأدوات التحكم الموجودة على النموذج. ويمكن الربط بين ملف مساعدة وبين مكون HelpProvider باستخدام خاصية HelpNamespace. ويتم تحديد نوع المساعدة المقدمة باستدعاء وسيلة SetHelpNavigator وتحديد قيمة من تعداد HelpNavigator للمتحكم المستهدف. ويتم تحديد الكلمة المفتاحية (Keyword) أو الموضوع (Topic) الخاص بالمساعدة عن طريق استدعاء وسيلة SetHelpKeyword.

وللربط بين سلسلة نص مساعدة معين وبين متحكم أخر، نستخدم وسيلة SetHelpString. والسلسلة التي يتم ربطها مع متحكم باستخدام هذه الوسيلة، تعرض في نافذة منطلقة (Pop-up window) عندما يضغط المستخدم على مفتاح F1 في حالة حيازة المتحكم لبؤرة التركيز. وعندما لا يتم ضبط خاصية HelpNamespace، يجب استخدام وسيلة SetHelpString لتوفير نص المساعدة. وعند ضبط كل من مكتبة المساعدة باستخدام خاصية HelpNamespace وسلسلة المساعدة، سوف يكون للمساعدة التي تعتمد على HelpNamespace الأولوية. وعندما يتم إضافة مكون HelpProvider إلى نموذج، فإنه يظهر

فى مربع المكونات أسفل مصمم نماذج الويندوز.

قائمة سرد الرسومات (ImageList)

يستخدم مكون ImageList فى عرض صور على أدوات تحكم، مثل متحكم ListView، متحكم TreeView، متحكم ToolBar، متحكم Button، ومتحكم TabControl. وتستخدم هذه القائمة فى تخزين الصور التى يمكن بعد ذلك عرضها بواسطة أدوات التحكم. وتسمح لنا قائمة الصور بكتابة كود خاص بكتالوج من الصور. على سبيل المثال، يمكن تغيير الصورة المعروضة على أحد الأزرار بتغيير قيمة خاصية ImageIndex. ويمكن أيضا الربط بين قائمة واحدة من الصور وبين العديد من أدوات التحكم فى نفس الوقت. على سبيل المثال، عندما نستخدم متحكم ListView ومتحكم TreeView لعرض نفس قائمة الملفات، فإن تغيير أيقونة أحد الملفات فى قائمة الأيقونات يترتب علية ظهور الأيقونة الجديدة فى كل من المشهدين. ويمكن استخدام قائمة الصور مع أى متحكم به خاصية ImageList، SmallImageList، LargeImageList. ولربط أحد أدوات التحكم مع مكون ImageList، نضبط خاصية ImageList بالمتحكم على اسم مكون ImageList.

والخاصية الأساسية فى مكون ImageList هى خاصية Images، التى تحتوى على الصور التى يتم استخدامها بواسطة أداة التحكم المرتبطة بالقائمة. ويمكن الوصول إلى كل صورة منفردة باستخدام الرقم الذى يمثلها فى فهرس الصور. وتحدد خاصية ColorDepth عدد الألوان التى تستخدمها الصور. وتعرض الصور بنفس الحجم الذى تحدده خاصية ImageSize. ويتم تغيير حجم الصور الكبيرة لتتناسب مع الحجم المحدد.

إضافة وحذف الصور فى مكون ImageList

يتم تأهيل مكون ImageList بالصور قبل ربطه مع متحكم. من ناحية أخرى، يمكن إضافة وحذف الصور إلى المكون بعد ربطة مع أحد أدوات التحكم. وعند حذف الصور، يجب التحقق من صلاحية خاصية ImageIndex فى أدوات التحكم المرتبطة مع مكون ImageList.

لإضافة أو حذف صور باستخدام المصمم:

١. نختار مكون ImageList أو نضيف واحدا إلى النموذج.

٢. فى نافذة Properties، ننقر على نقاط الحذف التالية لخاصية Images. يترتب على ذلك عرض نافذة Image Collection Editor.

٣. نستخدم أزرار Add و Remove لإضافة وحذف الصور من القائمة.

لإضافة صور باستخدام الكود، نستخدم وسيلة Add فى خاصية Images بمكون ImageList، كما يتضح من الكود التالى:

```
Dim myImage As System.Drawing.Image =
  Image.FromFile ("C:\winnt\Sample.gif")
ImageList1.Images.Add (myImage)
```

لحذف الصور باستخدام الكود، نستخدم وسيلة Remove لحذف صورة واحدة أو وسيلة Clear لحذف جميع الصور فى مكون ImageList، كما يتضح من الكود التالى:

```
ImageList1.Images.Remove (0)
ImageList1.Images.Clear ()
```

مكون القائمة الرئيسية (MainMenu Component)

يعرض هذا المكون قائمة فى وقت التشغيل. وعند إضافة هذا المركب إلى مصمم نماذج الويندوز، يسمح لنا مصمم القائمة بتكوين هيكل القائمة الرئيسية. وتعتبر كل القوائم الفرعية من القائمة الرئيسية وجميع البنود الموجودة بالقوائم، كائنات MenuItem. ويمكن تخصيص بند قائمة ليكون البند الإفتراضى عن طريق ضبط خاصية DefaultItem على القيمة True. ويظهر البند الإفتراضى فى بنط أسود عريض عند النقر على القائمة. ويمكن ضبط خاصية Checked فى بند القائمة على القيمة True أو القيمة False. وتقوم خاصية RadioCheck بضبط شكل ظهور البند الذى يتم اختياره. عندما تكون قيمة هذه الخاصية True، يظهر زر راديو بجانب البند. وعندما تكون القيمة False، تظهر علامة اختيار بجانبه. وقد تم شرح القوائم فى الفصل الثالث وما يتعلق بها موضوعات.

أيقونة الإشعار (NotifyIcon)

نستخدم مكون NotifyIcon لعرض أيقونات تمثل البرامج التى يجرى تنفيذها فى خلفية النظام ولاتعرض واجهة بينية (Interface) فى معظم الوقت. من الأمثلة على ذلك، برنامج الحماية من فيروسات الكمبيوتر الذى يمكن الوصول إلية بالنقر على أيقونة فى

شريط المهام (Task Bar). ويعرض كل مكون NotifyIcon أيقونة واحدة فى منطقة المهام بشريط الأوضاع. على هذا الأساس، إذا كان لدينا ثلاثة برامج عاملة فى الخلف ونريد عرض أيقونات لهم، يجب إضافة ثلاثة مكونات NotifyIcon إلى النموذج. وتشمل الخصائص الرئيسية فى هذا المكون خاصية Icon، وخاصية Visible. حيث تحدد خاصية Icon الأيقونة التى تظهر على شريط المهام. ولكى تظهر هذه الأيقونة، يجب ضبط خاصية Visible على القيمة True. ويمكن أن يرتبط بالأيقونات رسائل صغيرة وقوائم مختصرة (Context Menus).

إضافة أيقونات التطبيقات إلى شريط المهام باستخدام مكون NotifyIcon

يعرض مكون NotifyIcon أيقونة واحدة فى شريط المهام. ولعرض أيقونات متعددة، يجب أن يكون هناك عدد من مكونات NotifyIcon على النموذج. ولضبط الأيقونات المعروضة، نستخدم خاصية Icon. ويمكن أيضا كتابة كود فى معالج حدث DoubleClick لتنفيذ تعليمات عندما يقوم المستخدم بالنقر على أحد هذه الأيقونات. على سبيل المثال، يمكن عرض مربع حوار أمام المستخدم لتهيئة البرنامج العامل فى خلفية النظام، الذى تمثلة الأيقونة. ويمكن ضبط الأيقونة باستخدام نافذة Properties، أو باستخدام الكود.

لضبط أيقونة باستخدام نافذة Properties:

1. نـنقر عـلى نقاط الحذف التالية لخاصية Icon ثم نختار الملف الذى امتداده ico فى مربع حوار Open.
2. نضبط خاصية Visible على القيمة True.
3. نضبط خاصية Text على سلسلة نص قصيرة.

ويمكن تنفيذ ذلك بالكود التالى:

```
NotifyIcon1.Icon = New System.Drawing.Icon ("c:\antivirus.ico")
NotifyIcon1.Visible = True
NotifyIcon1.Text = "Antivirus program"
```

مربع حوار فتح الملفات (OpenFileDialog)

مكون OpenFileDialog هو مربع حوار سابق الإعداد. ويستخدم هذا المكون داخل تطبيقات الويندوز لتبسيط اختيار الملفات بدلا من تكوين مربعات حوار خاصة بنا. من

ناحية أخرى، يجب كتابة الكود اللازم لفتح هذه الملفات عند الحاجة إلى استخدامها. ونستخدم وسيلة ShowDialog لعرض مربع الحوار فى وقت التشغيل. ونستطيع تمكين المستخدمين من اختيار أكثر من ملف فى نفس الوقت باستخدام خاصية MultiSelect. بالإضافة إلى ذلك، يمكننا استخدام خاصية ShowReadOnly لعرض مربع اختيار القراءة فقط (Read-Only check box) فى مربع الحوار. وتشير خاصية ReadOnlyChecked إلى اختيار مربع القراءة فقط أو عدم اختياره. وأخيرا تقوم خاصية Filter بضبط سلسلة اختيار الملفات، التى تقرر بدورها الخيارات التى تظهر فى مربع Files of Type بمربع الحوار. ونستطيع إضافة مكون OpenFileDialog إلى النموذج عن طريق سحبة من مربع ToolBox. ويترتب على ذلك، إضافة المكون فى مربع المكونات أسفل مصمم النماذج.

المثال التالى يقوم بتكوين كائن من تصنيف OpenFileDialog، يضبط العديد من الخصائص، ويعرض مربع الحوار باستخدام وسيلة ShowDialog. ويفترض هذا المثال، وجود نموذج، ووجود متحكم Button على النموذج.

```
Protected Sub button1_Click (sender As Object, e As System.EventArgs)
   Dim myStream As Stream
   Dim openFileDialog1 As New OpenFileDialog ()

   openFileDialog1.InitialDirectory = "c:\"
   openFileDialog1.Filter = "txt files (*.txt)|*.txt|All files (*.*)|*.*"
   openFileDialog1.FilterIndex = 2
   openFileDialog1.RestoreDirectory = True

   If openFileDialog1.ShowDialog() = DialogResult.OK Then
      myStream = openFileDialog1.OpenFile()
      If Not (myStream Is Nothing) Then
         myStream.Close()
      End If
   End If
End Sub
```

مربع حوار ضبط صفحة الطباعة (PageSetUpDialog)

مكون PageSetUpDialog هو مربع حوار سابق الإعداد، يستخدم فى ضبط تفصيلات صفحة الطباعة فى تطبيقات الويندوز. ويمكن السماح للمستخدم بتعديل الحدود،

الهوامش، المقدمة والمؤخرة، وصورة طباعة الصفحة. ويترتب على استخدام مربع PageSetUpDialog وغيرة من مربعات الحوار القياسية، تكوين تطبيقات مألوفة للمستخدمين تتوافق مع معايير تطبيقات الويندوز. ولعرض مربع حوار ضبط صفحة الطباعة المذكور، نستخدم وسيلة ShowDialog في وقت التشغيل.

يحتوى هذا المكون على خصائص يمكن ضبطها وتتعلق بصفحة منفردة أو أى مستند منفرد. بالإضافة إلى ذلك، يمكن استخدام هذا المكون لتحديد قيم الضبط الخاصة بالطابعة وتخزينها فى تصنيف PrinterSettings. ويترتب على إضافة مكون PageSetUpDialog إلى النموذج، وضعة فى مربع المكونات أسفل مصمم النماذج.

مربع حوار الطباعة (PrintDialog)

مكون PrintDialog هو مربع حوار سابق الإعداد، يستخدم فى اختيار الطابعة، اختيار صفحات الطباعة، وتحديد قيم الضبط الأخرى الخاصة بالطباعة فى تطبيقات الويندوز. ويتيح هذا المكون للمستخدمين طباعة أجزاء متعددة من المستندات، مثل طباعة كامل الوثيقة، طباعة عدد من الصفحات، أو طباعة اختيار محدد. ولعرضْ مربع حوار الطباعة فى وقت التشغيل، نستخدم وسيلة ShowDialog.

ويحتوى هذا المكون على خصائص تتعلق بوظيفة طباعة واحدة، أو ضبط الطابعات. ويمكن المشاركة فى ذلك بواسطة طابعات متعددة. وعند إضافة هذا المكون إلى التطبيق، يظهر فى مربع المكونات أسفل مصمم النماذج.

مكون مستند الطباعة (PrintDocument Component)

يجرى استخدام مكون PrintDocument لضبط الخصائص التى تحدد متطلبات طباعة الوثيقة داخل تطبيقات الويندوز. ويمكن استخدام هذا المكون بالترافق مع مكون PrintDialog لتحقيق التحكم فى كل نواحي طباعة الوثيقة. وهناك نوعان رئيسيان من استخدامات مكون PrintDocument:

- وظائف الطباعة البسيطة، مثل طباعة ملف نص منفرد. فى هذه الحالة، نقوم بإضافة مكون PrintDocument إلى نموذج الويندوز، ثم نستخدم وسيلة Print لطباعة المستند فى داخل إجراء معالجة حدث PrintPage. تقوم هذه الوسيلة بإرسال كائن رسومات

تحتوى علية خاصية Graphics الموجودة فى تصنيف PrintPageEventArgs، إلى الطابعة.

- وظائف الطباعة الأكثر تعقيدا، مثل إعادة استخدام كود الطباعة الذى نقوم بكتابته. فى هذه الحالة، نقوم باشتقاق مكون جديد من مكون PrintDocument والهيمنة على حدث PrintPage.

استخدام مكون PrintDocument فى الطباعة

يمثل مكون PrintDocument التصنيف المستخدم للقيام بعملية الطباعة. وتمر عملية الطباعة بثلاث خطوات رئيسية:

١. نقوم بتكوين مثل جديد من مكون PrintDocument.

٢. نضبط الخصائص التى تصف ما سوف نقوم بطباعته باستخدام تصنيف PrinterSettings، وتصنيفPageSettings.

٣. نستخدم وسيلة Print لتنفيذ الطباعة.

وقد تم شرح الموضوعات المتعلقة بالطباعة مع الأمثلة الضرورية فى القسم الخاص بدعم الطباعة فى نماذج الويندوز.

مربع حوار حفظ الملفات (SaveFileDialog)

مكون SaveFileDialog هو مربع حوار سابق الإعداد. وعند استخدام هذا المكون، يجب كتابة الكود اللازم لحفظ الملفات. ولإظهار مربع حوار حفظ الملفات فى وقت التشغيل، نستخدم وسيلة ShowDialog. ويمكن فتح أحد الملفات للقراءة والكتابة باستخدام وسيلة OpenFile. ويظهر هذا المكون عند إضافته فى مربع المكونات أسفل مصمم نماذج الويندوز.

يشتمل المثال التالى على تكوين كائن من تصنيف SaveFileDialog، ضبط الخصائص، استدعاء مربع حوار حفظ الملفات باستخدام وسيلة ShowDialog، وفتح الملف الذى يتم اختياره. ويفترض المثال وجود نموذج به متحكم Button.

```
Protected Sub button1_Click (sender As Object, e As System.EventArgs)
    Dim myStream As Stream
    Dim saveFileDialog1 As New SaveFileDialog ()
```

```
    saveFileDialog1.Filter = "txt files (*.txt)|*.txt|All files (*.*)|*.*"
    saveFileDialog1.FilterIndex = 2
    saveFileDialog1.RestoreDirectory = True

    If saveFileDialog1.ShowDialog () = DialogResult.OK Then
        myStream = saveFileDialog1.OpenFile()
        If Not (myStream Is Nothing) Then
            myStream.Close()
        End If
    End If
End Sub
```

مكون المؤقت (Timer Component)

يؤدى مكون Timer إلى وقوع حدث على فترات متساوية منتظمة. وهذا المكون مصمم للاستخدام فى بيئة نماذج الويندوز. ويتم تحديد طول الفترة الفاصلة بين وقوع الأحداث باستخدام خاصية Interval، التى تتحدد قيمتها بجزء من الثانية. وعندما يكون هذا المكون متاحا، يقع حدث Tick كل فترة فصل. فى إجراء معالجة هذا الحدث، نضيف الكود الذى نريد تنفيذه. والخصائص الرئيسية فى مكون Timer هى خاصية Timer، خاصية Start، وخاصية Stop التى تحول المؤقت بين الفتح والإقفال.

استخدام الإجراءات مع مكون Timer

من الضروري أحيانا تكوين إجراء يعمل على فترات منفصلة إلى أن يتم الانتهاء من تنفيذ حلقة من الكود أو انتهاء فاصل زمنى. ويجعل مكون Timer مثل هذا الإجراء ممكنا. لتشغيل إجراء على فترات محددة باستخدام هذا المكون، نتبع الخطوات التالية:

١. نسحب مكون Timer من مربع ToolBox إلى النموذج.

٢. نضبط خاصية Intervals. هذه الخاصية، تحدد الوقت الذى سوف يمر قبل تشغيل الإجراء مرة أخرى.

٣. نكتب الكود المناسب فى إجراء معالجة حدث Tick. سوف يتكرر تشغيل هذا الكود عند انتهاء الفترة الفاصلة المحددة.

٤. فى الوقت المناسب، نضبط خاصية Enabled على القيمة False لإيقاف تشغيل الإجراء. ويجب الانتباه إلى أن ضبط الفترة الفاصلة على صفر لا يؤدى إلى إيقاف

مكون Timer.

أمثلة على استخدام مكون Timer

يتتبع المثال التالى وقت اليوم بزيادة ثانية فى كل مرة. ويستخدم متحكم Button، متحكم Label، ومكون Timer على نموذج. ويتم ضبط خاصية Intervals لكى تساوى ١٠٠٠ وهو المعادل لواحد ثانية. وفى إجراء معالجة حدث Tick، يتم ضبط عنوان متحكم Label على الوقت الحالى. وعند النقر على متحكم Button، يجرى ضبط خاصية Enabled على القيمة False لإيقاف عمل المؤقت ومنع تحديث عنوان متحكم Label.

```
Private Sub InitializeTimer ()
  Timer1.Interval = 1000
  Timer1.Enabled = True
  Button1.Text = "Enabled"
End Sub

Private Sub Timer1_Tick (ByVal Sender As Object, ByVal e As EventArgs) Handles
Timer1.Tick
  Label1.Text = DateTime.Now
End Sub

Private Sub Button1_Click (ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    If Button1.Text = "Stop" Then
      Button1.Text = "Start"
      Timer1.Enabled = False
    Else
      Button1.Text = "Stop"
      Timer1.Enabled = True
    End If
End Sub
```

وفى المثال الثانى يتم تشغيل إجراء كل ٢٠٠٠ مللي ثانية إلى أن يتم الخروج من حلقة

```
Private counter As Integer
Private Sub InitializeTimer ()
  Counter = 1
  Timer1.Interval = 2000
  Timer1.Enabled = True
End Sub
```

```
Private Sub Timer1_Tick (ByVal sender As Object, ByVal e As System.EventArgs)
Handles Timer1.Tick
  If counter > 10 Then
    Timer1.Enabled = False
    Counter = 1
  Else
    MessageBox.Show (counter)
    Counter = counter + 1
  End If
End Sub
```

مكون المعلومات المختصرة (ToolTip Component)

يعرض هذا المكون نصا عندما يشير المستخدم إلى أداة تحكم. ويمكن ربط مكون ToolTip مع أى متحكم. من أمثلة استخدام هذا المكون، عرض أيقونة صغيرة على زر وربطة مع مكون ToolTip لشرح وظيفة هذا الزر. ويمكن استخدام نسخة واحدة من هذا المكون مع عدد من أدوات التحكم فى نفس الوقت. والوسائل الرئيسية فى مكون ToolTip هى: وسيلة SetToolTip المستخدمة فى ضبط المعلومة التى يتم عرضها، ووسيلة GetToopTip التى تمكننا من الحصول على المعلومة التى يحتوى عليها هذا المكون. والخصائص الرئيسية هى: خاصية Active، التى يجب ضبطها على القيمة True لكى تظهر المعلومة التى يحتوى عليها المكون، وخاصية AutomaticDelay التى تضبط الوقت الذى يستغرقه عرض المعلومة، الوقت الذى يجب أن ينتظره المستخدم عند وضع مؤشر الماوس على أحد أدوات التحكم قبل ظهور هذه المعلومة، والوقت الواجب انتظاره قبل ظهور المعلومة مرة أخرى.

ضبط وقت التأخير فى مكون ToolTip

. ووحدة القياس ToolTipهـناك العديـد من قيم التأخير التى يمكن ضبطها بخصوص مكون الوقت InitialDelayالمستخدمة فـى تحديـد هـذه القيم هـى مللـي ثانـية. تحـدد خاصـية المسـتغرق تـند الإشـارة إلى مـتحكم قـبل ظهـور المعلومـة المختصـرة. وتضـبط خاصـية عـدد وحـدات مللى ثانية المستغرقة قبل إعادة عرض المعلومة عندما يتحرك ReshowDelay طول وقت عرض سلسلة AutoPopDelayالماوس من متحكم إلى متحكم أخر. وتحدد خاصية . يمكـن ضـبط هـذه الخصـائص عـلى انفـراد، أو ضـبط قـيمة خاصـية ToolTipمعلومـة الـتى تقوم بدورها بضبط قيم التأخير الأخرى على أساس نسبة ثابتة من AutomaticDelay

على AutomaticDelay. عـلى سبيل المثال، عند ضبط قيمة خاصية AutomaticDelayقـيمة ، ضـبط خاصـية N عـلى القـيمة InitialDelay مـن الوقـت، يـتم ضـبط خاصـية Nالقـيمة 10N . على القيمة AutoPopDelay، وضبط خاصية N/5 على القيمة ReshowDelay

لضبط التأخير، نضبط الخصائص التالية باستخدام الكود:

```
ToolTip1.InitialDelay = 500
ToolTip1.ReshowDelay = 100
ToolTip1.AutoPopDelay = 5000
```

ويمكن ضبط معلومة ToolTip الخاصة بأدوات التحكم باستخدام الكود:

١. نضيف المتحكم الذى سوف يعرض معلومة ToolTip.

٢. نستخدم وسيلة SetToolTip لتحديد المعلومة.

```
ToolTip1.SetToolTip (Button1, "Save changes")
```

لضبط معلومة ToolTip باستخدام مصمم النماذج:

١. نضيف المتحكم الذى سوف يعرض معلومة ToolTip.

٢. نضيف مكون ToolTip إلى النموذج.

٣. نختار المتحكم الذى سوف يعرض معلومة ToolTip، أو نضيفة إلى النموذج.

٤. فى نافذة Properties، نضبط قيمة ToolTip On ToolTip1 على النص المناسب.

تطبيق استخدام الرسوم المتحركة

يوضح هذا التطبيق كيفية تنفيذ تحريك الرسوم باستخدام +GDI. شاملا حركة عين إنسانية، حركة كرة ترتد عند الاصطدام بحواجز، ورسم نص باستخدام تحريك الألوان. ويشتمل التطبيق على الكائنات التالية:

- نموذج رئيسي (Main Form) لرسم وتحرك الرسومات.
- ثلاثة من أزرار الراديو (Radio Buttons) لاختيار الرسم الذى نريد عرضة.
- قائمة (Menu) لاستخدامها في إغلاق التطبيق.

فيما يلى الخطوات التنفيذية للتطبيق:

١. إضافة مراجع إلى مناطق الأسماء التي نحتاج إليها.

```
Option Strict On
Imports System.Drawing.Drawing2D
Imports System.Drawing.Text

Public Class frmMain
   Inherits System.Windows.Forms.Form
```

٢. إجراء معالجة اختيار بند الخروج من القائمة.

```
Private Sub mnuExit_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles mnuExit.Click
      ' Close the current form
      Me.Close()
End Sub
```

٣. الإعلان عن ثابت يمثل الفترة الفاصلة بين تكرار رسومات العين.

```
Const WINK_TIMER_INTERVAL As Integer = 150 ' In milliseconds
```

٤. الإعلان عن مصفوفة تحتوى على عدد أربعة رسومات للعين

```
Protected arrImages(4) As Image
```

٥. الإعلان عن متغير يمثل الرسم المعروض حاليا.

```
Protected intCurrentImage As Integer = 0
Protected j As Integer = 1
```

٦. الإعلان عن ثابت يمثل الفترة التي تفصل بين تكرار رسومات الكرة.

```
Const BALL_TIMER_INTERVAL As Integer = 25 ' In milliseconds
```

٧. الإعلان عن متغير حجم الكرة على أنها جزء من حجم منطقة الرسم.

```
Private intBallSize As Integer = 16
```

٨. الإعلان عن متغير مسافة حركة الكرة على أساس أنها جزء من حجم الكرة.

```
Private intMoveSize As Integer = 4
```

٩. الإعلان عن متغير رسم الكرة.

```
Private bitmap As bitmap
```

١٠. الإعلان عن متغيرات إحداثيات موقع الكرة فى منطقة الرسم.

```
Private intBallPositionX, intBallPositionY As Integer
```

١١. الإعلان عن متغيرات إحداثيات نصف قطر الكرة، إحداثيات موقع حركة الكرة، عرض الرسم، وارتفاع الرسم.

```
Private intBallRadiusX, intBallRadiusY, intBallMoveX, intBallMoveY, _
        intBallBitmapWidth, intBallBitmapHeight As Integer
```

١٢. الإعلان عن متغيرات هوامش العرض والطول للرسم الذى يمثل الكرة.

```
Private intBitmapWidthMargin, intBitmapHeightMargin As Integer
```

١٣. الإعلان عن ثابت يمثل الفترة الفاصلة بين تكرار كتابة النص.

```
Const TEXT_TIMER_INTERVAL As Integer = 15 ' In milliseconds
```

١٤. الإعلان عن متغير يمثل درجة تغيير الميل الجاري.

```
Protected intCurrentGradientShift As Integer = 10
```

١٥. الإعلان عن متغير يمثل خطوة تغيير الميل.

```
Protected intGradiantStep As Integer = 5
```

١٦. استخدام إجراء تحميل النموذج فى تعبئة مصفوفة رسوم العين بكائنات الرسم. ويجب تواجد ملفات الرسم بالمسار الموجود فى الكود التالى أو تغيير المسار للوصول إلى تلك الملفات.

```
Private Sub frmMain_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    ' Fills the image array for the Winking Eye example.
    Dim i As Integer
    For i = 0 To 3
        arrImages(i) = New bitmap("..\Eye" & (i + 1).ToString & ".png")
    Next i
End Sub
```

١٧. الإجراء التالي يقوم بتغيير قيمة فترة تكرار الرسم بناء على الرسم الذى يتم اختياره بواسطة أزرار الراديو ثم يستدعى إجراء تغيير حجم النموذج. .

```
Private Sub RadioButtons_CheckedChanged(ByVal sender As Object, ByVal e As System.EventArgs) Handles optWink.CheckedChanged, optBall.CheckedChanged
    If optWink.Checked Then
        tmrAnimation.Interval = WINK_TIMER_INTERVAL
    ElseIf optBall.Checked Then
        tmrAnimation.Interval = BALL_TIMER_INTERVAL
    ElseIf optText.Checked Then
        tmrAnimation.Interval = TEXT_TIMER_INTERVAL
    End If
    OnResize(EventArgs.Empty)
End Sub
```

١٨. الإجراء التالى يعالج حدث Tick فى مكون المؤقت (Timer Component). ويتم

داخل هذا الإجراء تنفيذ الرسومات المتحركة.

```
Protected Overridable Sub TimerOnTick(ByVal obj As Object, ByVal ea As
EventArgs) Handles tmrAnimation.Tick

    If optWink.Checked Then
```

١٩. الحصول على كائن تنفيذ الرسومات الذى يمثل صفحة الرسم.

```
        Dim grfx As Graphics = CreateGraphics()
```

٢٠. استدعاء وسيلة تنفيذ الرسم وتمرير كائن الرسم، إحداثيات موقع الرسم فى مركز النموذج، عرض وارتفاع الرسم.

```
        grfx.DrawImage(arrImages(intCurrentImage), _
            CInt((ClientSize.Width - arrImages(intCurrentImage).Width) / 2), _
            CInt((ClientSize.Height - arrImages(intCurrentImage).Height) / 2), _
            arrImages(intCurrentImage).Width, _
            arrImages(intCurrentImage).Height)
```

٢١. التخلص من كائن تنفيذ الرسومات.

```
        grfx.Dispose()
```

٢٢. الحركة بين الرسومات فى حلقة.

```
        intCurrentImage += j
        If intCurrentImage = 3 Then
```

٢٣. عند الوصول إلى الرسم الأخير نعكس ترتيب الرسومات لكى نبدأ من جديد ويتم إقفال العين.

```
            j = -1
        ElseIf intCurrentImage = 0 Then
```

٢٤. عند الوصول إلى الرسم الأول، نعكس ترتيب الرسومات لكى يتم فتح العين.

```
            j = 1
        End If

    ElseIf optBall.Checked Then
```

٢٥. تكوين كائن رسومات جديد لاستخدامة فى رسم الكرة المتحركة.

```
    Dim grfx As Graphics = CreateGraphics()
    grfx.DrawImage(bitmap, _
            CInt(intBallPositionX - intBallBitmapWidth / 2), _
            CInt(intBallPositionY - intBallBitmapHeight / 2), _
            intBallBitmapWidth, intBallBitmapHeight)
```

```
grfx.Dispose()
```

٢٦. زيادة إحداثيات موقع الكرة على أساس إحداثيات الحركة الجديدة.

```
intBallPositionX += intBallMoveX
intBallPositionY += intBallMoveY
```

٢٧. تغيير اتجاه الكرة عند اصطدامها بأحد الموانع.

```
If intBallPositionX + intBallRadiusX >= ClientSize.Width _
   Or intBallPositionX - intBallRadiusX <= 0 Then
   intBallMoveX = -intBallMoveX
   Beep()
End If
```

٢٨. ضبط الإحداثي Y لكي لا ترتد الكرة إلى داخل أدوات التحكم الموجودة على النموذج.

```
If intBallPositionY + intBallRadiusY >= ClientSize.Height _
   Or intBallPositionY - intBallRadiusY <= 40 Then
   intBallMoveY = -intBallMoveY
   Beep()
End If

ElseIf optText.Checked Then
```

٢٩. الحصول على لوحة الرسم لتنفيذ كتابة النصوص.

```
Dim grfx As Graphics = CreateGraphics()
```

٣٠. ضبط طاقم الحروف المستخدم، النص الذى سوف يتم عرضة، والحجم.

```
Dim font As New font("Microsoft Sans Serif", 96, _
   FontStyle.Bold, GraphicsUnit.Point)
Dim strText As String = "GDI+!"
Dim sizfText As New SizeF(grfx.MeasureString(strText, font))
```

٣١. ضبط نقطة بدء رسم النص فى مركز منطقة الرسم.

```
Dim ptfTextStart As New PointF( _
   CSng(ClientSize.Width - sizfText.Width) / 2, _
   CSng(ClientSize.Height - sizfText.Height) / 2)
```

٣٢. ضبط نقاط بدء ونهاية الميل، ويتم تغيير نقطة النهاية لكي تتحقق الحركة.

```
Dim ptfGradientStart As New PointF(0, 0)
Dim ptfGradientEnd As New PointF(intCurrentGradientShift, 200)
```

٣٣. تكوين الفرشاة المستخدمة فى رسم النص.

```
    Dim grBrush As New LinearGradientBrush(ptfGradientStart, _
      ptfGradientEnd, Color.Blue, BackColor)
```

٣٤. رسم النص فى مركز منطقة الرسم.

```
    ' Draw the text centered on the client area.
    grfx.DrawString(strText, font, grBrush, ptfTextStart)

    grfx.Dispose()
```

٣٥. تحريك الميل ثم عكس اتجاهه عندما يصل إلى قيمة محددة.

```
    intCurrentGradientShift += intGradiantStep
    If intCurrentGradientShift = 500 Then
      intGradiantStep = -5
    ElseIf intCurrentGradientShift = -50 Then
      intGradiantStep = 5
    End If
  End If
End Sub
```

٣٦. الوسيلة التالية تهيمن على وسيلة OnResize فى التصنيف الأصلى. وتقوم بمعالجة حدث تغيير الحجم.

```
Protected Overrides Sub OnResize(ByVal ea As EventArgs)
  If optWink.Checked Then
```

٣٧. الحصول على لوحة الرسم وإزالة الرسومات الموجودة.

```
    Dim grfx As Graphics = CreateGraphics()
    Me.Refresh()
    grfx.Dispose()

  ElseIf optBall.Checked Then
    Dim grfx As Graphics = CreateGraphics()
    grfx.Clear(BackColor)
```

٣٨. ضبط نصف قطر رسم الكرة ليكون جزءا من عرض أو ارتفاع منطقة الرسم أيهما أقل.

```
    Dim dblRadius As Double = Math.Min(ClientSize.Width / grfx.DpiX, _
      ClientSize.Height / grfx.DpiY) / intBallSize
```

٣٩. ضبط عرض وارتفاع الكرة.

```
    intBallRadiusX = CInt(dblRadius * grfx.DpiX)
    intBallRadiusY = CInt(dblRadius * grfx.DpiY)
```

```
grfx.Dispose()
```

٤٠. ضبط مسافة حركة الكرة ليكون واحد بكسل أو جزء من حجم الكرة أيهما أكبر. يعنى ذلك أن المسافة التى تتحركها الكرة تتناسب مع حجمها، ويكون ذلك بدورة متناسبا مع حجم منطقة الرسم. يترتب على ذلك بط حركة الكرة عند انكماش منطقة الرسم والعكس صحيح.

```
intBallMoveX = CInt(Math.Max(1, intBallRadiusX / intMoveSize))
intBallMoveY = CInt(Math.Max(1, intBallRadiusY / intMoveSize))
intBitmapWidthMargin = intBallMoveX
intBitmapHeightMargin = intBallMoveY
```

٤١. تحديد حجم Bitmap الفعلى المستخدم فى رسم الكرة بإضافة الهوامش إلى أبعاد الكرة.

```
intBallBitmapWidth = 2 * (intBallRadiusX + intBitmapWidthMargin)
intBallBitmapHeight = 2 * (intBallRadiusY + intBitmapHeightMargin)
```

٤٢. تكوين رسم جديد وتمرير العرض والارتفاع.

```
bitmap = New bitmap(intBallBitmapWidth, intBallBitmapHeight)
```

٤٣. الحصول على لوحة الرسم، حذف رسم الكرة، ثم إعادة رسم الكرة مرة أخرى.

```
grfx = Graphics.FromImage(bitmap)
With grfx
    .Clear(BackColor)
    .FillEllipse(Brushes.Red, New Rectangle(intBallMoveX, _
       intBallMoveY, 2 * intBallRadiusX, 2 * intBallRadiusY))
    .Dispose()
End With
```

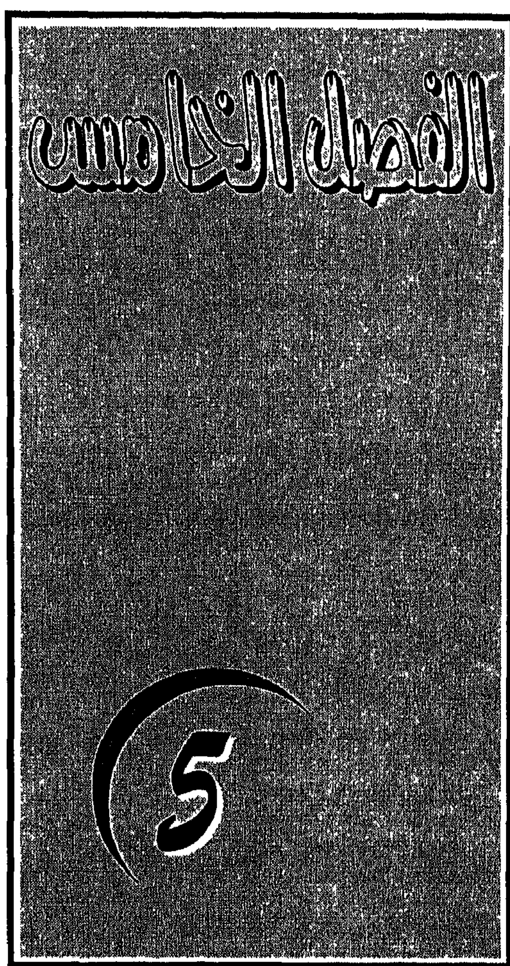
٤٤. إعادة ضبط موقع الكرة على مركز منطقة الاستخدام فى النموذج.

```
intBallPositionX = CInt(ClientSize.Width / 2)
intBallPositionY = CInt(ClientSize.Height / 2)

ElseIf optText.Checked Then
```

٤٥. الحصول على لوحة الرسم وحذف الرسومات الموجودة.

```
Dim grfx As Graphics = CreateGraphics()
grfx.Clear(BackColor)
End If
End Sub
End Class
```



قواعد البيانات

تدور معظم تطبيقات Visual Basic حول قراءة وتحديث المعلومات الموجودة فى قواعد البيانات. ولتحقيق هذا الهدف يوجد العديد من تقنيات الوصول إلى البيانات التى قدمتها شركة مايكروسوفت، مثل تقنية DAO، تقنية ADO، وأخيرا قدمت مايكروسوفت تقنية ADO.NET الجديدة للعمل فى بيئة الإنترنت. ولتحقيق تكامل البيانات فى التطبيقات الموزعة، توفر بيئة Visual Studio الدعم لهذا الجيل الجديد من تكنولوجيا الوصول إلى البيانات الممثل فى ADO.NET.

تقنية ADO.NET

هناك أهداف مختلفة للتطبيقات التى تستخدم ADO.NET فى التعامل مع البيانات. قد يكون الهدف من هذه التطبيقات هو عرض البيانات فى أحد النماذج. وفى حالات أخرى، يمكن أن يكون الهدف من هذه التطبيقات هو مشاركة البيانات مع أطراف أخرى. وبغض النظر عن الهدف من تطبيقات البيانات، يجب فهم هيكل التعامل مع البيانات فى ADO.NET والمكونات الشائعة به. ولذلك سوف نستعرض هنا أهم المفاهيم المتعلقة بهذه التقنية.

١. لا تعتمد ADO.NET على الاتصالات الحية المستمرة على عكس المعمول به فى تطبيقات الخادم والعميل (Client/Server) التقليدية. ويرجع السبب فى ذلك إلى أن الاتصالات المستمرة تستهلك جزءا مهما من موارد النظام. ويترتب عليها تدنى أداء هذه التطبيقات. ولذلك، تم تصميم تقنية الوصول إلى البيانات فى ADO.NET بالطريقة التى تؤدى إلى الاقتصاد فى الاتصالات. وأصبح الاتصال بين التطبيقات وبين قواعد البيانات يتم لفترة تكفى فقط للحصول على البيانات أو تحديثها. وبسبب عدم الاحتفاظ باتصالات عاطلة مع قاعدة البيانات لفترة طويلة، يمكن أن تخدم قاعدة البيانات فى ADO.NET الكثير من المستخدمين.

٢. تنفيذ التفاعلات مع قاعدة البيانات يتم باستخدام أوامر SQL أو الإجراءات المخزنة التى تحتوى بدورها على أوامر SQL. يتم ذلك فى ADO.NET، عن طريق تكوين أوامر البيانات (Data Commands) التى تحتوى على عبارات SQL أو الإجراءات

المخزنة التى نريد تنفيذها. يلى ذلك فتح اتصال مع قاعدة البيانات، تنفيذ عبارة SQL أو الإجراء المخزن، ثم إقفال الاتصال. ويمكن أن تشتمل أوامر البيانات على معاملات تتيح لنا التحكم فى طريقة التعامل مع قاعدة البيانات أثناء التشغيل.

٣. البيانات التى نحصل عليها من قاعدة البيانات، يمكن تخزينها فى ذاكرة وسيطة بالكمبيوتر يطلق عليها فئة بيانات (dataset). إن الذهاب إلى قاعدة البيانات فى كل مرة يحتاج فيها التطبيق إلى معالجة السجل التالى فى جدول بيانات، لا يكون عمليا فى حالات كثيرة. والحل الأمثل لهذة المشكلة يكمن فى تخزين السجلات المستخرجة من قاعدة البيانات بصورة مؤقتة والعمل مع هذه الفئة المؤقتة من البيانات. وهذا هو ما تقوم به فئة البيانات (dataset). وتشتمل فئة البيانات على واحد أو أكثر من الجداول التى تقابل الجداول الموجودة فى قاعدة البيانات الحقيقية. كما يمكن أن تشتمل على علاقات بين الجداول وعلى قيود مفروضة على البيانات الموجودة فى هذه الجداول. وتعتبر البيانات الموجودة فى فئة البيانات نسخة مختصرة من تلك الموجودة فى قاعدة البيانات ويمكن العمل معها بطريقة مشابهة إلى حد كبير للطريقة التى نتعامل بها مع البيانات الحقيقية. ويكون الاتصال مقطوعا مع قاعدة البيانات أثناء العمل مع فئة البيانات ثم يتم تحديث قاعدة البيانات بمحتويات فئة البيانات فى فترة لاحقة. ونظرا لأن فئة البيانات تعتبر نسخة خاصة من قاعدة البيانات، لذلك لا تعكس فئة البيانات بالضرورة الوضع الجاري فى قاعدة البيانات. وعندما نريد الحصول على أخر التغييرات التى تمت فى قاعدة البيانات بواسطة المستخدمين، يجب تجديد فئة البيانات بإعادة قراءة قاعدة البيانات مرة أخرى.

٤. تعمل فئة البيانات مستقلة عن قاعدة البيانات. ولهذا السبب، تعتبر فئة البيانات نقطة تكامل جيدة للبيانات القادمة من مصادرة متعددة. على سبيل المثال، يمكن أن تأتى البيانات من قاعدة بيانات ويأتي جزء أخر من قاعدة بيانات أخرى أو من مصدر أخرى غير قواعد البيانات. وبمجرد أن تصبح البيانات فى فئة البيانات، يمكننا العمل معها بغض النظر عن المصدر الأصلى للبيانات.

٥. الصيغة المستخدمة لتحويل البيانات من مخزن بيانات إلى فئة بيانات، ومن فئة بيانات إلى المكونات الأخرى فى نظام ADO.NET هى صيغة XML. وعند الرغبة فى حفظ البيانات فى ملف، يتم حفظها فى صيغة XML. على هذا الأساس، يمكننا استخدام ملفات XML مصدرا للبيانات واستخدامها لإنشاء فئات البيانات. وتعتبر XML الصيغة الأساسية للبيانات فى نظام ADO.NET. ويقوم هذا النظام أتوماتيكيا بإنشاء ملفات XML وتحويل البيانات الخاصة بفئة البيانات إلى تيار XML وإرسالها إلى مكون أخر. ويقوم المكون الأخر بتحويل تيار XML الوارد إلى بيانات ووضعها مرة أخرى فى فئة بيانات. ويجب الإنتباة إلى أن البيانات لا يتم تخزينها فى صيغة XML بفئة البيانات. ويترتب على استخدام XML فى بناء بروتوكولات البيانات، توفر كثير من المزايا، مثل إمكانية تبادل بيانات أحد التطبيقات مع أى تطبيق أو مكون فى أى تطبيق أخر، تستخدم XML النصوص فى تمثيل البيانات ولا تستخدم النظام الثنائى لهذا يمكن إرسالها من خلال أى بروتوكول مثل HTTP ، كما أنة ليس من الضروري معرفة كود XML لأن ADO.NET تقوم تلقائيا بتحويل البيانات من والى XML.

٦. استخدام XML فى تصميم مخططات (Schemas) هياكل البيانات التى تحتوى عليها فئات البيانات. بموجب ذلك، يتم تعريف الجداول، الأعمدة، أنواع البيانات، القيود المفروضة، وغيرها فى فئات البيانات باستخدام مخططات XML. وفى معظم الحالات، لا نحتاج إلى التعمق فى مخططات فئات البيانات لأن Visual Studio.NET يقوم بتكوين وتحديث المخططات عند الحاجة بناءا على ما نقوم به باستخدام أدوات التصميم المرئية. على سبيل المثال، عند استخدام الأدوات المرئية لتكوين فئة بيانات، يقوم Visual Studio بتوليد مخططات XML التى تصف هياكل فئة البيانات. ويتم بعد ذلك استخدام هذا المخطط فى توليد فئة بيانات نوعية تحتوى على جداول، أعمدة، وغيرها من عناصر هياكل البيانات.

توريد البيانات فى نظام NET Framework

يقوم مورد البيانات فى نظام NET Framework بوظيفة الجسر بين تطبيق وبين مصدر

بيانات. ويستخدم لاستخلاص البيانات من مصدر البيانات ونقل التغييرات التى تحدث بهذه البيانات فى الاتجاه المعاكس إلى مصدر البيانات. ويشتمل نظام NET Framework على كائنات توريد البيانات التالية:

- SQL Server .NET Data Provider. يعمل هذا المورد مع Microsoft SQL Server الإصدار رقم ٧ ومع الإصدارات التى بعدة.
- OLE DB .NET Data Provider. يعمل مع مصادر البيانات التى تتعامل مع OLE DB.
- ODBC .NET Data provider. هذا المورد لا يتم تحميلة تلقائيا عند تثبيت نظام NET Framework، ولكن يمكن تحميلة منفصلا.

ويتكون مورد البيانات (Data Provider) فى نظام NET Framework من الكائنات الأساسية التالية:

- كائن Connection. وهو الكائن الذى يقوم ببناء اتصال مع مصدر بيانات محدد.
- كائن Command. هو الكائن الذى ينفذ أمر معين على مصدر بيانات.
- كائن DataReader. يقرأ تيار من البيانات القابلة للقراءة فقط من مصدر بيانات.
- كائن DataAdapter. يقوم بتأهيل فئة بيانات ونقل تحديث البيانات بها إلى مصدر البيانات.

بالإضافة إلى التصنيفات الأساسية السابقة المعروضة أعلاه، يحتوى نموذج مورد البيانات فى NET Framework على التصنيفات التالية:

- كائن Transaction. يتيح لنا تنفيذ الأوامر فى مجموعات على مصدر البيانات.
- كائن CommandBuilder. هو كائن مساعد يقوم تلقائيا ببناء الأمر فى مورد البيانات أو يشتق معلومات المعاملات من إجراء مخزن ويؤهل مجموعة معاملات كائن الأمر.
- كائن Parameter. يحدد الإدخالات، المخرجات، ويعيد معاملات للأوامر والإجراءات المخزنة.

- كائن Exception. يتم الحصول علية عند حدوث خطأ فى مصدر البيانات.
- كائن Error. يعرض المعلومات العائدة من مصدر بيانات، المتعلقة بتحذير أو خطأ.
- كائن ClientPermission. يختص بمتطلبات السرية المتعلقة بالوصول إلى كود مورد بيانات NET.

تكوين الاتصال مع مصادر البيانات

لتحريك البيانات بين مخزن بيانات وبين التطبيق، يجب أولا بناء اتصال مع مخزن البيانات (Data Store). ويمكن بناء وإدارة اتصال فى ADO.NET باستخدام واحد من كائنات الاتصال التالية:

- كائن SqlConnection الذى يقوم بإدارة اتصال مع خادم SQL الإصدار ٧ أو ما بعدة.
- كائن OleDbConnection الذى يقوم بإدارة اتصال مع أى مخزن بيانات ممكن الوصول إلية من خلال OLE DB.

سلاسل الاتصال

أهم خاصية فى كائن الاتصال هى خاصية سلسلة الاتصال (ConnectionString)، التى تحتوى على المعلومات المطلوبة للدخول على قاعدة البيانات. ويمكن ضبط خاصية سلسلة الاتصال (ConnectionString) باعتبارها سلسلة واحدة، أو بضبط خصائص الاتصال كل على حدة. ويمكن أن تحتوى هذه الخاصية على سلسلة مماثلة للسلسلة التالية:

```
Provider=SQLOLEDB.1;Data Source=MySQLServer;Initial
Catalog=NORTHWIND;Integrated Security=SSPI
```

تحتوى هذه السلسلة على اسم مورد البيانات (Provider)، مصدر البيانات (Data Source)، اسم قاعدة البيانات (Catalog)، ونظام الأمان المستخدم (Integrated Security)، وهو نظام أمان الويندوز (NT authentication). ويمكن أن تحتوى السلسلة على كود تعريف المستخدم وعلى كلمة المرور.

ويمكن ضبط خاصية ConnectionString عندما يكون الاتصال مقفلا فقط. ويترتب على إعادة ضبط سلسلة اتصال إعادة ضبط جميع قيم الاتصال بما فيها كلمة المرور. على

سبيل المثال، عند ضبط سلسلة اتصال تحتوى على "Initial Catalog=northwind"، ثم إعادة الضبط لكى تشتمل السلسلة على "DataSource = myserver; Password = Hh78relv"، فإن خاصية Database لن تحتوى على northwind. ومن الملاحظ هنا أن خاصية Initial Catalog هى نفسها خاصية Database. وتتم مراجعة سلسلة الاتصال مبدئيا عند ضبط القيم بها. على سبيل المثال، عندما تحتوى السلسلة على قيمة Provider، قيمة Connect Timeout، قيمة Persist Security Info، فإن هذه القيم يتم تحقيقها عند تكوين السلسلة. وعند استدعاء وسيلة Open فى كائن السلسلة، يجرى تحقيق كامل سلسلة الاتصال. ويتم إصدار رسالة خطأ عندما تحتوى السلسلة على خصائص غير صالحة أو غير مدعمة. ويمكن الفصل بين القيم باستخدام علامات الاقتباس المفردة أو المزدوجة. ويجب الفصل بين الأزواج التى تتكون من الكلمات المرشدة والقيم الخاصة بها باستخدام الفاصلة المنقوطة.

المثال التالى يفترض وجود متحكم Button1 على سطح نموذج، ويوضح كيفية تكوين كائن اتصال من نوع OleDbConnection وكيفية ضبط بعض الخصائص فى سلسلة الاتصال:

```
Imports System.Data
Imports System.Data.OleDb
Public Class Form1
   Inherits System.Windows.Forms.Form
#Region " Windows Form Designer generated code "
   Private Sub Button1_Click (ByVal sender As System.Object, ByVal e
      As System.EventArgs) Handles Button1.Click
      CreateOleDbConnection ()
   End Sub
   Public Sub CreateOleDbConnection ()
      Dim myConnString As String = _
        "Provider=SQLOLEDB;Data Source=FHOME1;Initial" & _
         "Catalog=Northwind;Integrated Security=SSPI;"
      Dim myConnection As New OleDbConnection (myConnString)
      myConnection.Open ()
      MessageBox.Show ("ServerVersion: " +
      myConnection.ServerVersion _
        + ControlChars.Cr + "DataSource: " +
      myConnection.DataSource.ToString())
      myConnection.Close ()
   End Sub
```

End Class

فتح وإقفال الاتصالات

وأهم الوسائل فى كائن الاتصال، وسيلة Open ووسيلة Close. تستخدم وسيلة Open المعلومات المتوفرة فى خاصية ConnectionString لبناء الاتصال مع مصدر البيانات. وتقوم وسيلة Close بإغلاق الاتصال. ويعتبر إغلاق الاتصال ضروريا لأن معظم مصادر البيانات تدعم عددا محددا من الاتصالات المفتوحة، كما أن الاتصالات المفتوحة تستنفذ موارد مهمة فى النظام.

وعند استخدام موفقات البيانات (Data Adapters) أو أوامر البيانات (Data Commands)، لا يكون من الضروري فتح وإقفال الاتصالات صراحة. لأن استدعاء بعض الوسائل الخاصة بهذه الكائنات، مثل وسيلة Fill أو وسيلة Update بكائن موفق البيانات يترتب علية قيام هذه الوسائل بفحص الاتصال. فإذا لم يكن مفتوحا، يتم فتحة والقيام بالعمليات المستهدفة ثم إقفال الاتصال مرة أخرى.

المشاركة فى الاتصالات

يجب المشاركة في الاتصالات (Pooling Connections) بين المستخدمين الذين يقومون باستعلام نفس قاعدة البيانات للحصول على نفس البيانات لأن السماح لكل مستخدم بإجراء اتصال منفصل يمكن أن يكون له تأثير سلبي على الأداء. ويقوم مورد البيانات بمعالجة المشاركة فى الاتصال تلقائيا عند استخدام تصنيفOleDbConnection فى الاتصال، كما يقوم مورد البيانات بنفس العمل عند استخدام تصنيف SqlConnection مع توفير خيار إدارة المشاركة فى الاتصال بواسطة المستخدم.

خصائص الاتصال القابلة للإعداد

قد لا يكون متاحا معرفة معلومات الاتصال فى وقت التصميم، مثل اسم الخادم، اسم المستخدم، وكلمة المرور. ولذلك، يتم فى الغالب تعريف سلاسل الاتصال على أنها خصائص ديناميكية. وبالنظر إلى أن الخصائص الديناميكية يتم تخزينها فى ملف إعداد (Configuration File) ولا يتم ترجمتها ضمن الملفات الثنائية الخاصة بالتطبيق. لذلك يمكن تغييرها بدون الحاجة إلى إعادة ترجمة التطبيق. كما يمكن بعد ذلك توفير طريقة أمام

المستخدمين للقيام فى وقت التشغيل بتحديث ملف الإعداد لكى يقوم نظام NET Framework بالقراءة منة وتنفيذ الاتصال على أساسه.

تأمين معلومات الاتصال

ونظرا لأهمية الموارد التى يتم الوصول إليها عن طريق الاتصالات، لذا يتم توفير السرية لهذة الاتصالات عن طريق خصائص السرية المتوفرة فى نظام الويندوز. وفى التطبيقات التى تتكون طبقتين (Two-tier Applications)، تجرى الاستفادة من نظام سرية الويندوز ونظام السرية الموجود فى نظام SQL Server.

اتصالات وقت التصميم

وبالإضافة إلى الاتصال مع مصادر البيانات فى وقت التشغيل، نحتاج أيضا للوصول إلى هذه المصادر فى وقت التصميم. ولهذا يوفر مربع Server Explorer فى Visual Studio طريقة لبناء اتصالات مع مصادر البيانات فى وقت التصميم. مما يوفر إمكانية استعراض مصادر البيانات المتاحة، عرض المعلومات عن الجدول والأعمدة والعناصر الأخرى التى تحتويها هذه المصادر. وتستخدم المعلومات المتوفرة عن اتصالات وقت التصميم فى ضبط خصائص كائن اتصال بغرض إضافته إلى التطبيق، تصميم النماذج عن طريق استعراض قاعدة البيانات وسحب أعمدة الجداول إلى النموذج، نسخ معلومات سلسلة الاتصال باستخدام اتصال وقت التصميم.

تكوين كائنات الاتصال

من المعتاد ألا نحتاج مباشرة إلى تكوين وإدارة كائنات الاتصال فى Visual Studio لأن الأدوات التى نستخدمها، مثل Data Adapter Wizard، تقوم أتوماتيكيا بتكوين كائنات الاتصال على النموذج الذى نستخدمه. ومع ذلك، يمكننا إضافة كائنات اتصال بأنفسنا إلى نموذج وضبط خصائصه. وهناك عدد من الخيارات المتاحة لتكوين كائن اتصال، تتمثل فيما يلى:

- استخدام أحد أدوات التصميم التالية التى تقوم بتكوين كائن اتصال على أساس أنة جز من المهمة التى تقوم بها هذه الأدوات:

- DataAdapter Configuration Wizard. تقوم هذه الأداة بقيادة المستخدم للحصول على المعلومات اللازمة لتكوين اتصال يرتبط مع موفق بيانات.
- Data Form Wizard. تقوم هذه الأداة بتكوين كائن اتصال على أساس أنة جزء من النموذج الذى نقوم بتهيئته.

- سحب جدول، أعمدة، أو إجراء مخزن من مربع Server Explorer إلى النموذج. يترتب على سحب أحد هذه العناصر إلى النموذج، تكوين موفق بيانات وكائن اتصال.
- إنشاء اتصال مستقل (Stand Alone Connection). يترتب على ذلك إنشاء كائن اتصال على النموذج. ونقوم بتهيئة خصائص هذا الكائن يدويا. وتعتبر هذه الاستراتيجية مفيدة عند الرغبة فى ضبط خصائص الاتصال وقت التشغيل أو عند ضبط الخصائص باستخدام نافذة Properties.

لتكوين اتصال مستقل باستخدام الأدوات المرئية:

١. من ملصق Data بمربع الأدوات، نسحب كائن اتصال إلى النموذج. نستخدم كائن SqlConnection للاتصال مع SQL Server الإصدار ٧ أو ما بعدة. ونستخدم كائن OleDbConnection للاتصال مع مصدر بيانات أخر أو إذا كان هناك احتمال تغيير مصدر البيانات فى المستقبل.
٢. نختار كائن الاتصال فى المصمم ثم نستخدم نافذة Properties لضبط كائن الاتصال.
   أ. يمكن ضبط خاصية ConnectionString باعتباره وحدة واحدة.
   ب. أو يمكن ضبط كل خاصية على حدة.
٣. نضبط خاصية Name عند الرغبة فى إعادة تسمية الاتصال.
٤. عندما نريد ضبط الخصائص وقت التشغيل بدون إعادة ترجمة البرنامج، نقوم بتحديد خصائص الكائن على أنها ديناميكية.

كائن الامر (Command Object)

بعد بناء اتصال مع مصدر بيانات، يمكن تنفيذ أوامر واستعادة نتائج من مصدر البيانات باستخدام كائن الأمر. يمكن تكوين كائن أمر باستخدام إجراء بناء الأمر، الذى يستقبل معاملات اختيارية تتكون من عبارة SQL لتنفيذها على مصدر بيانات، كائن اتصال (Connection Object)، كائن عملية (Transaction Statement). ويمكن أيضا تكوين أمر خاص باتصال محدد باستخدام وسيلة CreateCommand الموجودة فى كائن الاتصال. ويمكن الاستعلام عن وتعديل عبارة SQL الخاصة بالأمر عن طريق استخدام خاصية CommandText. ويعرض كائن الأمر عدة نسخ من وسائل Execute التى يمكن استخدامها لتنفيذ الأعمال المطلوبة. عند إعادة تيار من البيانات، يمكن استخدام وسيلة ExecuteReader لإعادة كائن DataReader. ونستخدم وسيلة ExecuteScalar لإعادة قيمة منفردة، ونستخدم وسيلة ExecuteNonQuery لتنفيذ أوامر لا تعيد سجلات بيانات.

وعند استخدام كائن Command مع أحد الإجراءات المخزنة، يمكن ضبط خاصية CommandType فى كائن Command على StoredProcedure. استخدام قيمة StoredProcedure مع خاصية CommandType فى الأمر، يتيح استخدام خاصية Parameters فى كائن Command للوصول إلى معاملات الإدخال والإخراج والقيم العائدة. ويمكن الوصول إلى خاصية Parameters بغض النظر عن وسيلة Execute المنفذة. ومع ذلك، عند استدعاء وسيلة ExecuteReader، لن يمكن الوصول إلى القيم العائدة فى معاملات الإخراج إلى أن يتم إقفال كائن قراءة البيانات (DataReader). يوضح الكود التالى كيفية تكوين كائن بيانات يعيد قائمة تحتوى على مجموعات من قاعدة بيانات Northwind الموجودة فى SQL Server.

عند استخدام مورد بيانات SQL Server

```
Dim catCMD As SqlCommand = New SqlCommand ("SELECT CategoryID,
CategoryName FROM Categories", myConnection)
```

عند استخدام مورد بيانات OleDb

```
Dim catCMD As OleDbCommand = New OleDbCommand ("SELECT CategoryID,
CategoryName FROM Categories", myConnection)
```

كائن قارئ البيانات (DataReader)

يمكن استخدام كائن DataReader للحصول على تيار بيانات من قاعدة بيانات يقبل القراءة فقط في اتجاه واحد. ويمكن أن يترتب على استخدام كائن DataReader زيادة فاعلية أدوات التطبيقات وتخفيض أعباء النظام بسبب وجود صف واحد فقط من البيانات فى الذاكرة. ويأتي تكوين كائن DataReader بعد تكوين كائن Command عن طريق استدعاء وسيلة Command.ExecuteReader لاستخراج الصفوف من مصدر البيانات، كما يتضح من المثال التالى الذى يستخدم أمر بيانات المثال السابق:

```
Dim myReader As SqlDataReader = catCMD.ExecuteReader ()
```

ونستخدم وسيلة Read فى كائن DataReader للحصول على صف بيانات من نتائج استعلام. ويمكن الوصول إلى كل عمود فى ذلك الصف عن طريق تمرير اسم أو ترتيب العمود إلى كائن DataReader. من ناحية أخرى، يوفر لنا كائن DataReader مجموعة من الوسائل التى تمكننا من الوصول إلى قيم الأعمدة على أساس أنواع بياناتها الأصلية، مثل GetDateTime، GetDouble، وغيرها. المثال التالى يقوم باستخراج عمودين من كل صف يتم قراءته قارئ البيانات myReader السابق تكوينه:

```
Do While myReader.Read ()
  Console.WriteLine (vbTab & "{0}" & vbTab & "{1}", myReader.GetInt32 (0),
  myReader.GetString (1))
Loop
myReader.Close ()
```

ويسمح كائن DataReader بالمعالجة المتسلسلة للنتائج التى يتم الحصول عليها من مصدر البيانات. ويعتبر هذا الكائن خيارا جيدا عند استخراج كميات كبيرة من البيانات لأن البيانات لا يتم وضعها فى ذاكرة وسيطة بالذاكرة.

إغلاق كائن DataReader

يجب دائما استدعاء وسيلة Close عند الانتهاء من استخدام كائن DataReader. وإذا كان كائن Command يحتوى على معاملات إخراج أو قيم عائدة، لن يكون فى الإمكان التعامل معها إلى أن يتم الانتهاء من إغلاق كائن DataReader. ويجب ملاحظة أن كائن DataReader يستخدم الاتصال بصورة كاملة إلى أن يتم إغلاق ذلك الكائن. يعنى ذلك، أننا

لا نستطيع استخدام هذا الاتصال لتنفيذ أوامر أخرى طالما ظل كائن DataReader مفتوحا.

تعدد مجموعات السجلات العائدة من تنفيذ كائن DataReader

في هذه الحالة يوفر كائن DataReader وسيلة NextResult لتكرار المعالجة بالنسبة للمجموعات المختلفة من السجلات بالتسلسل، كما يتضح من المثال التالي:

```
Imports System.Data
Imports System.Data.SqlClient
Public Class Form1
   Inherits System.Windows.Forms.Form
#Region " Windows Form Designer generated code "
   Private Sub Button1_Click (ByVal sender As System.Object, ByVal e As
   System.EventArgs) Handles Button1.Click
         CreateMyConnection ()
   End Sub
   Public Sub ReadMRecords ()
      Dim myCMD As SqlCommand = New SqlCommand ("SELECT CategoryID,
         CategoryName FROM Categories;" & _
         "SELECT EmployeeID, LastName FROM Employees", nwindConn)
      nwindConn.Open ()
      Dim myReader As SqlDataReader = myCMD.ExecuteReader ()
      Dim fNextResult As Boolean = True
      Do Until Not fNextResult
         Console.WriteLine (vbTab & myReader.GetName (0) & vbTab & _
         myReader.GetName (1))
         Do While myReader.Read ()
            Console.WriteLine (vbTab & myReader.GetInt32 (0) & vbTab & _
             myReader.GetString (1))
         Loop
         fNextResult = myReader.NextResult ()
      Loop
      myReader.Close ()
      nwindConn.Close ()
   End Sub
End Class
```

الحصول على معلومات مخطط البيانات من كائن *DataReader*

أثناء فتح كائن DataReader، يمكننا استخراج معلومات مخطط البيانات الخاص بمجموعة السجلات الجارية باستخدام وسيلة GetSchemaTable. تعيد لنا هذه الوسيلة

كائن DataTable مؤهل بالصفوف والأعمدة التى تحتوى على معلومات المخطط لمجموعة الصفوف الحالية. وسوف يحتوى كائن DataTable على صف لكل عمود من أعمدة مجموعة البيانات الجارية. وكل عمود فى صف جدول المخطط يقابل خاصية بالعمود العائد ضمن مجموعة سجلات البيانات الجارية. الكود التالى يكتب معلومات مخطط البيانات التى يعيدها كائن DataReader.

```
Imports System.Data
Imports System.Data.SqlClient
Public Class Form1
   Inherits System.Windows.Forms.Form
#Region " Windows Form Designer generated code "
   Private Sub Button1_Click (ByVal sender As System.Object, ByVal e
   As System.EventArgs) Handles Button1.Click
      CreateMyConnection ()
   End Sub
   Public Sub CreateMyConnection ()
      Dim myCMD As SqlCommand = New SqlCommand ("SELECT CategoryID,
      CategoryName FROM Categories;" & _
         "SELECT EmployeeID, LastName FROM Employees", nwindConn)
      nwindConn.Open ()
      Dim myReader As SqlDataReader = myCMD.ExecuteReader ()
      Dim schemaTable As DataTable = myReader.GetSchemaTable ()
      Dim myRow As DataRow
      Dim myCol As DataColumn
      For Each myRow In schemaTable.Rows
         For Each myCol In schemaTable.Columns
            Console.WriteLine (myCol.ColumnName & " = " &
            myRow (myCol).ToString ())
         Next
         Console.WriteLine ()
      Next
      myReader.Close ()
      nwindConn.Close ()
   End Sub
End Class
```

كائنات موفقات البيانات (Data Adapters)

كائن موفق البيانات هو أحد التصنيفات التى يحتوى عليها نموذج مورد البيانات فى نظام NET Framework. وتستخدم الموفقات فى تبادل البيانات بين مصدرها وبين فئات البيانات (datasets). يعنى ذلك فى الكثير من التطبيقات، قراءة البيانات من قاعدة البيانات ووضعها فى فئة البيانات، ثم كتابة تغييرات البيانات من فئة البيانات إلى قاعدة البيانات. يقدم لنا Visual Studio اثنين من موفقات البيانات الأساسية للاستخدام مع قواعد البيانات:

- كائن OleDbDataAdapter الذى يستخدم مع أى مصدر بيانات يتيحه مورد بيانات OLE DB.
- كائن SqlDataAdapter المستخدم مع مورد بيانات SQL Server الإصدار رقم ٧ وما بعدة.

ويمكن إتاحة أنواع أخرى من موفقات البيانات غير النوعين السابقين بواسطة أطراف أخرى قابلة للتكامل مع Visual Studio.NET. ويقوم كل موفق بتبادل البيانات بين جدول واحد فى مصدر بيانات وبين كائن DataTable فى فئة بيانات. وإذا كانت فئة البيانات تحتوى على أكثر من جدول، فإن الإستراتيجية المعتادة هى تكوين العديد من موفقات البيانات التى تقوم بتأهيل كائنات الجداول بالبيانات ثم كتابة تغييرات البيانات التى تحدث فى كائنات الجداول بالجداول المقابلة فى قواعد البيانات. وعندما نريد تأهيل جدول فى فئة بيانات، نقوم باستدعاء وسيلة فى موفق البيانات لتنفيذ عبارة SQL أو إجراء مخزن (Stored Procedure). ويقوم موفق البيانات بتكوين كائن قارئ بيانات OleDbDataReader، أو SqlDataReader لقراءة البيانات ووضعها فى فئة البيانات. وعندما نريد تحديث قاعدة البيانات، نقوم باستدعاء وسيلة فى موفق بيانات لتنفيذ عبارة SQL مناسبة أو إجراء مخزن للقيام بعملية التحديث الفعلى لقاعدة البيانات.

وظائف موفقات البيانات

يقوم موفق البيانات بعدد من الوظائف، بعضها وظائف رئيسية والأخرى وظائف

مساعدة. تتمثل الوظائف الرئيسية في تأهيل فئات البيانات من قواعد البيانات وتحديث قواعد البيانات بالتغييرات التي تحدث في هذه الفئات. وتشمل الوظائف المساعدة إجراء اتصال مع قاعدة البيانات، تنفيذ أوامر البيانات على قواعد البيانات بعد تمرير المعاملات المطلوبة، وتحقيق التناظر بين الجداول الموجودة في قاعدة البيانات وتلك الموجودة في فئة البيانات.

اتصال موفقات البيانات مع قواعد البيانات

يحتاج موفق البيانات إلى فتح اتصال مع مصدر البيانات لقراءة وكتابة البيانات المختلفة. لتحقيق ذلك، يستخدم موفق البيانات كائنات الاتصال من نوع OleDbConnection أو SqlConnection للاتصال مع مصادر البيانات من هذا النوع. ويحتوى موفق البيانات على أربعة مراجع اتصال، مرجع خاص بفعل Insert، فعل Update، فعل Select، وفعل Delete. ويمثل كائن الاتصال فترة تحاور (Session) مستقلة عن فترات التحاور الأخرى داخل مصدر البيانات. ويوفر كل من كائن OleDbConnection وكائن SqlConnection خصائص لتعديل الاتصال، مثل خاصية تعريف المستخدم، خاصية تعريف كلمة المرور، وخاصية فترة الاتصال. كما يوفر وسائل لبدء، تنفيذ، وحذف عمليات قاعدة البيانات.

استخدام كائنات الأوامر في موفقات البيانات

باستخدام موفق البيانات يمكن قراءة، إضافة، تحديث، وحذف السجلات في مصدر بيانات. وللسماح لنا بتحديد كيفية عمل كل واحدة من هذه العمليات، يدعم موفق البيانات الخصائص الأربعة التالية:

- خاصية SelectCommand التي تحتفظ بمرجع إلى عبارة SQL أو الإجراء المخزن الذي يستخلص الصفوف من مخزن البيانات.
- خاصية InsertCommand التي تحتفظ بمرجع إلى أمر إدراج الصفوف في مخزن بيانات.
- خاصية UpdateCommand التي تحتفظ بمرجع إلى أمر تعديل الصفوف في

مخزن بيانات.

- خاصية DeleteCommand التى تحتفظ بمرجع إلى أمر حذف الصفوف من مخزن بيانات.

وتعتبر هذه الخصائص كائنات من تصنيف OleDbCommand أو SqlCommand. وتدعم هذه الكائنات خاصية CommandText التى تحتوى على مرجع إلى عبارة SQL أو إجراء مخزن. ويجب ملاحظة أن تصنيف Command يجب أن يتوافق مع تصنيف Connection. على سبيل المثال، عند استعمال كائن SqlConnection للاتصال مع SQL Server، يجب استخدام أوامر مشتقة من تصنيف SqlCommand أيضا. ومع أنة يمكننا صراحة ضبط نص OleDbCommand، أو SqlCommand، فإننا لا نحتاج دائما إلى فعل ذلك. لأن موفق البيانات يمكن أن يقوم تلقائيا بإنتاج عبارات SQL المناسبة فى وقت التشغيل. من ناحية أخرى، يمكن معالجة كائنات Command أثناء وقت التصميم وأثناء وقت التشغيل مما يمكن من التحكم المباشر فى كيفية تنفيذ هذه الأوامر. على سبيل المثال، يمكننا إنشاء أو تعديل الأمر المرتبط مع كائن SelectCommand قبل تنفيذه مباشرة. ويمكننا أيضا تنفيذ الأوامر بأنفسنا بعيدا عن موفق البيانات.

استخدام معاملات الأوامر فى موفقات البيانات

فى موفق البيانات، تقود المعاملات تنفيذ الأوامر. على سبيل المثال، أمر خاصية SelectCommand، يحتوى فى الغالب على معامل فى فقرة WHERE لكى يمكن تحديد السجلات التى نريد استخراجها فى وقت التشغيل من قاعدة البيانات. وتستخدم الأوامر الأخرى المعاملات التى تسمح لنا فى وقت تشغيل التطبيقات، بتمرير البيانات التى نريد كتابتها فى سجل وتحديد السجل الذى نريد تحديثه فى قاعدة البيانات.

القراءة والتحديث فى موفقات البيانات

الغرض الأساسى من استخدام موفق البيانات هو توصيل البيانات بين مخزن بيانات وبين فئة بيانات. ويدعم الموفق وسائل محددة لتحريك البيانات للخلف والأمام بين الكائنين. ويجب الإنتباة إلى أنة إذا كان الهدف هو قراءة البيانات فقط، لا تكون هناك ضرورة لتخزين البيانات فى فئة بيانات. وبديلا عن ذلك، يمكن قراءة هذه البيانات مباشرة

من قاعدة البيانات إلى التطبيق. وللقيام بتحريك البيانات بين المصدر وبين فئة البيانات، يقوم موفق البيانات بتنفيذ العمليات التالية:

- استخراج الصفوف من مخزن بيانات إلى الجداول المقابلة فى فئة البيانات. ولاستخراج صفوف ووضعها فى فئة بيانات، نستخدم وسيلة Fill فى كائن SqlDataAdapter أو كائن OleDbDataAdapter. وعند استدعاء هذه الوسيلة، تقوم بإرسال عبارة SQL SELECT إلى مخزن البيانات.
- إرسال التغييرات التى حدثت فى جداول فئة البيانات إلى مخزن البيانات المقابل. ولإرسال جدول بفئة البيانات إلى مخزن بيانات، نستخدم وسيلة Update بموفق البيانات. وعند استدعاء هذه الوسيلة، تقوم بتنفيذ عبارة SQL المناسبة، مثل أمر UPDATE، أمر INSERT، أو أمر DELETE على أساس حالة السجل ذات العلاقة، سجل جديد، سجل معدل، أو سجل محذوف.

المناظرة بين جداول البيانات فى موفقات البيانات

عندما نستخدم أدوات Visual Studio لإنتاج فئة بيانات من جداول قاعدة بيانات، فإن الجداول والأعمدة فى فئة البيانات تستخدم نفس أسماء الأعمدة والجداول فى قاعدة البيانات المقابلة. قد تكون هذه الطريقة غير عملية فى بعض الحالات، مثل استخدام أسماء موجزة، طويلة، أو أجنبية فى قاعدة البيانات. وعند العمل مع مخطط (Schema) قائم من قبل، يمكن أن تكون الأسماء المستعملة مختلفة عن تلك الموجودة فى قاعدة البيانات. ولذلك، لا يكون من الضروري تطابق الأسماء المستخدمة فى فئة بيانات مع تلك المستخدمة فى قاعدة البيانات. وبديلا عن ذلك، يمكن إنشاء أسماء جديدة للجداول والأعمدة فى فئة البيانات، ثم مناظرة هذه الأسماء مع تلك المستخدمة فى قاعدة البيانات. للقيام بعملية المناظرة، تستخدم موفقات البيانات مجموعة باسم TableMappings للحفاظ على التناظر بين هياكل فئة البيانات المثلة فى الجداول والأعمدة وبين هياكل مخازن البيانات.

تكوين موفقات البيانات

فى Visual Studio، هناك عدد من الطرق التى يمكن استخدامها لتكوين وتهيئة موفقات البيانات. تشمل هذه الطرق: استخدام Server Explorer، استخدام أدوات

Wizards، استخدام نافذة Properties، أو استخدام الكود.

تكوين موفقات البيانات باستخدام *Server Explorer*

لتكوين موفق بيانات (Data Adapter) باستخدام Server Explorer، نطبق الخطوات التالية:

١. نفتح النموذج فى المصمم المناسب، ثم نفتح Server Explorer.

٢. نقوم بفتح اتصال، إذا لم يكن موجودا.

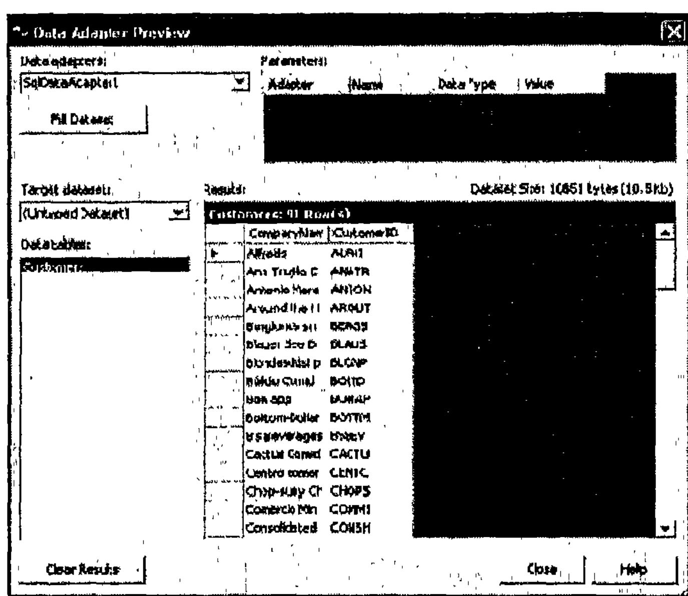
٣. نفتح عقدة Connection ونقوم بتوسع عقدة من العقد التالية: Tables، Stored Procedure، Views، أو Functions. وفى حالة العمل مع الجداول، نقوم بتوسيع عقدة اسم الجدول لعرض الأعمدة التى يحتوى عليها.

٤. نختار عمود أو أكثر من الأعمدة الموجودة تحت Table أو Stored Procedure ثم نسحب العمود الذى تم اختياره إلى المصمم. ويمكن سحب اسم الجدول إلى المصمم بدلا من اختيار وسحب الأعمدة المنفردة. يترتب على القيام بذلك، تكوين استعلام:

SQL SELECT * FROM tablename

ويعتبر اختيار أعمدة فرعية أكثر فاعلية بسبب تحديد كمية البيانات التى يتم تحويلها بين قاعدة البيانات وبين فئة البيانات. وعندما نقوم بوضع الاختيار على المصمم، يقوم Visual Studio بإنشاء مثل من كائن الاتصال مع مصدر البيانات ومثل من كائن موفق البيانات. وعندما نسحب جدول أو أعمدة مستقلة إلى المصمم، يتم تهيئة الموفق لكى يقوم بقراءة وتحديث البيانات. وعند سحب إجراء مخزن إلى المصمم، يتم تهيئة الموفق لقراءة البيانات فقط.

٥. فى حالة الرغبة فى مشاهدة كيفية قيام موفق البيانات بتأهيل فئة البيانات، نختار Preview Data من قائمة Data بالقائمة الرئيسية. يترتب على ذلك عرض مربع حوار Data Adapter Preview، المبين بالشكل رقم (١٣). نختار Data Adapter فى مربع السرد المركب أعلى يسار المربع، ثم ننقر زرFill Dataset لمشاهدة البيانات القادمة من قاعدة البيانات إلى فئة البيانات.



شكل رقم ١٣

ومع أن استخدام طريق سحب البيانات من Server Explorer سهل الاستخدام، إلا أنة لا يوفر الفرصة لإنشاء استعلام يحتوى على معاملات (Parameterized Query) أو إنشاء إجراء مخزن جديد للوصول إلى البيانات. لتحقيق ذلك، يمكن استخدام أداة Data Adapter Configuration Wizard.

تكوين موفقات البيانات باستخدام أدوات Wizard

توفر لنا هذه الأدوات أكثر الطرق سهولة ومرونة لإنشاء موفقات البيانات. ولتنفيذ هذه الطريقة:

١. نفتح النموذج أو المكون فى المصمم المناسب.

٢. من صفحة ملصق Data فى مربع ToolBox، نسحب كائن OleDbDataAdapter أو نسحب كائن SqlDataAdapter إلى وجه المصمم. يؤدى ذلك إلى قيام المصمم بإضافة مثل من الموفق إلى النموذج وعرض مربع حوار Data Adapter Configuration Wizard.

٣. فى مربع الحوار السابق، نقوم بتنفيذ ما يلى:

أ. فى الصفحة الثانية، نقوم بإنشاء أو اختيار اتصال.

ب. فى الصفحة الثالثة، نحدد ما إذا كان الموفق سوف يستخدم عبارات SQL أو الإجراءات المخزنة لقراءة وتحديث البيانات باستخدام واحدا من الخيارات المذكورة فى الجدول رقم (٢٠):

الاختيار	الإيضاح
Use SQL statement	نحدد عبارة SQL SELECT التى سوف يستخدمها الموفق لتأهيل جدول فى فئة بيانات. وبناءا على عبارة SELECT، سوف يقوم الموفق أيضا بإنتاج عبارات، UPDATE و عبارة DELETE لتحديث مصدر البيانات .
Use newly created stored procedures	نحدد عبارة SELECT، وسوف يقوم Wizard بتكوين إجراءات مخزنة من العبارة لقراءة وتحديث مصدر البيانات. وإذا لم يكن الموفق مدعما لهذا الخيار، لن يكون متاحا.
Use existing stored procedures	نحدد الإجراءات المخزنة الموجودة التى سوف يقوم الموفق باستخدامها لقراءة وتحديث مصدر البيانات.

جدول ٢٠

٤. فى الصفحة الرابعة، إما أن نقوم بإنشاء عبارة SELECT أو نختار إجراء مخزن. وللمساعدة فى بناء عبارة SQL، ننقر SQL Builder لبدء برنامج Query Builder.

عند الانتهاء من استخدام أداة Wizard، سوف تقوم هذه الأداة بتكوين مثل من كائن اتصال باستخدام المعلومات التى سبق تحديدها فى الصفحة الثانية. وعندما نرغب فى تغيير اسم الموفق واسم الاتصال، نختارهما على انفراد فى المصمم ثم نحدد لكل واحد منهما اسما فى

نافذة الخصائص. ونستخدم المشاهدة المبدئية لرؤية البيانات التى سوف يقوم الموفق بوضعها فى فئة البيانات.

تكوين موفقات البيانات باستخدام نافذة Properties

يمكن أيضا تكوين موفق البيانات بدويا. للقيام بذلك، يجب أن يكون لدينا كائن اتصال متاح. وتشتمل عملية الإعداد على تحديد المناظرة بين الجداول فى مصدر البيانات وفى فئة البيانات.

لإنشاء موفق البيانات يدويا، نتبع الخطوات التالية:

١. نتحقق من وجود كائن اتصال متاح بالنموذج الذى نعمل به.

٢. من صفحة ملصق Data بمربع ToolBox، نسحب كائن OleDataAdapter إلى وجه المصمم. يقوم المصمم بإضافة مثل من الموفق إلى النموذج وبدء برنامج Data Adapter Configuration Wizard.

٣. نقوم بإقفال مربع حوار Data Adapter Configuration Wizard.

٤. نختار الموفق، ثم نحدد الأوامر المستعملة لقراءة وتحديث البيانات باستخدام نافذة Properties. يجب تهيئة كائن SelectCommand. وإذا كان الموفق سوف يستخدم أيضا فى تحديث مصدر بيانات، يجب أيضا تهيئة الكائنات الأخرى: كائن InsertCommand، كائن UpdateCommand، كائن DeleteCommand. ولكل كائن من كائنات الأوامر، نقوم بضبط الخصائص المبينة بالجدول رقم (٢١).

الخاصية	الإيضاح
ActiveConnection	تحتوى على كائن اتصال. ويمكن تكوين كائن اتصال من هذه الخاصية فى نافذة Properties.
CommandText	تحتوى على عبارة SQL أو اسم إجراء مخزن.
CommandType	تحتوى على نوع الأمر الموجود فى خاصية CommandText.
Parameters	مجموعة من الكائنات من نوع Parameter المستخدمة لتمرير

الخاصية	الإيضاح
	القيم.

جدول ٢١

٥. وعندما لا نرغب فى استخدام نفس اسم العمود الموجود فى مصدر البيانات لعمود فئة البيانات، نقوم بتغيير القيم المتناظرة.

٦. من قائمة Data فى القائمة الرئيسية، نختار Generate Dataset. ويجب ملاحظة أن هذه القائمة لا تكون متاحة إلا عندما يحوز النموذج بؤرة التركيز.

٧. عند الرغبة فى مشاهدة كيفية تأهيل فئة البيانات بالمعلومات، يمكن استخدام Preview Data من قائمة Data بالقائمة الرئيسية.

تأهيل فئة بيانات باستخدام موفق بيانات

يمكن تعريف فئة البيانات (dataset) بأنها تمثيل مقيم بالذاكرة للبيانات، ومستقل عن مصدر هذه البيانات. وتمثل فئة البيانات مجموعة كاملة من البيانات التى تشتمل على جداول، قيود، وعلاقات بين هذه الجداول. ونظرا لأن فئة البيانات مستقلة عن مصدر البيانات، لذا يمكن أن تشتمل على بيانات محلية خاصة بالتطبيق، أو بيانات من مصادر متعددة. ويتم التحكم فى التعامل مع مصادر البيانات عن طريق استخدام موفقات البيانات، التى تقوم باستخراج المعلومات من قاعدة البيانات وتأهيل جداول فئات البيانات بها ثم نقل التغييرات الحادثة فى فئات البيانات إلى مصادر هذه البيانات. ويستخدم موفق البيانات كائنات الأوامر لاستخراج البيانات من مصدر بيانات وتأهيل جداول فئة بيانات بها، كما يستخدم كائنات الأوامر أيضا لنقل التغييرات الحادثة فى فئة البيانات إلى مصدر البيانات. وتشمل كائنات الأوامر التى يستخدمها موفق البيانات: كائن SelectCommand لاستخراج البيانات من مصدر البيانات، كائن UpdateCommand،كائن InsertCommand، وكائن DeleteCommand لتحديث البيانات فى مصدر البيانات. ولتحقيق هذه المهام، يقوم موفق البيانات بالاتصال مع مصدر البيانات عن طريق استخدام أحد كائنات الاتصال (Connection Object).

وتستخدم وسيلة Fill فى موفق البيانات لتأهيل فئة البيانات بنتائج الأمر الذى يحتوى

علية SelectCommand. وتتطلب وسيلة Fill تمرير اسم فئة البيانات المطلوب تأهيلها، وكائن DataTable إليها. وبدورها تستخدم وسيلة Fill كائن DataReader ضمنيا للحصول على أسماء الأعمدة وأنواعها المستخدمة فى إنشاء الجداول فى فئة البيانات، وبالمثل البيانات المستخدمة فى تأهيل صفوف الجداول فى فئة البيانات. ويتم إنشاء الجداول والأعمدة فقط إذا لم تكن موجودة، وإلا فإن وسيلة Fill تستخدم مخطط فئة البيانات الموجود. ويتم تكوين أنواع الأعمدة باستخدام الأنواع الموجودة فى نظام NET Framework ولا يتم إنشاء المفاتيح الرئيسية إلا إذا كانت موجودة فى مصدر البيانات. وإذا وجدت وسيلة Fill أن هناك مفاتيح أساسية بأحد الجداول، فإنها تقوم بكتابة البيانات الواردة من المصدر على بيانات أعمدة فئة البيانات التى تماثل قيم المفاتيح الأساسية بها تلك الموجودة بالأعمدة المستخرجة من مصدر البيانات. وإذا لم يكن هناك مفتاح أساسى، يتم إلحاق البيانات بالجداول فى فئة البيانات. وتستخدم وسيلة Fill أى مناظرة جدول (Table Mapping) موجودة عند تأهيل فئة البيانات.

المثال التالى، يقوم بإنشاء موفق بيانات يستخدم اتصال مع قاعدة بيانات Northwind فى Microsoft SQL Server لتأهيل كائن DataTable فى فئة بيانات بقائمة العملاء الموجودة فى قاعدة البيانات.

- استخدام موفق SqlDataAdapter:

```
Dim nwindConn As SqlConnection = New SqlConnection ("Data Source=localhost;Integrated Security=SSPI;Initial Catalog=northwind")


Dim selectCMD As SqlCommand = New SqlCommand ("SELECT CustomerID, CompanyName FROM Customers", nwindConn)
selectCMD.CommandTimeout = 30


Dim custDA As SqlDataAdapter = New SqlDataAdapter
custDA.SelectCommand = selectCMD
nwindConn.Open ()

Dim custDS As DataSet = New DataSet
custDA.Fill (custDS, "Customers")
nwindConn.Close ()
```

- استخدام موفق OleDb:

```
Dim nwindConn As OleDbConnection = New OleDbConnection ("Provider =
SQLOLEDB;Data Source=localhost;" & _
"Integrated Security=SSPI;Initial Catalog=northwind")

Dim selectCMD As OleDbCommand = New OleDbCommand ("SELECT CustomerID,
CompanyName FROM Customers", nwindConn)
selectCMD.CommandTimeout = 30

Dim custDA As OleDbDataAdapter = New OleDbDataAdapter
custDA.SelectCommand = selectCMD

Dim custDS As DataSet = New DataSet
custDA.Fill (custDS, "Customers")

nwindConn.Close ()
```

يلاحظ أن هذا الكود لم يقم صراحة بفتح وإقفال الاتصال. لأن وسيلة Fill تقوم ضمنيا بفتح الاتصال الذى يستخدمه موفق البيانات إذا وجدت أن الاتصال غير مفتوح. وإذا قامت وسيلة Fill بفتح الاتصال فإنها سوف تقوم أيضا بإغلاقه عند الانتهاء من مهمتها. يؤدى ذلك إلى تبسيط الكود المستخدم عند التعامل مع عملية واحدة، مثل Fill أو Update. من ناحية ثانية، عند استخدام عمليات متعددة تتطلب فتح اتصال، يمكن تحسين أداء التطبيق عن طريق فتح الاتصال صراحة، القيام بالعمليات مع مصدر البيانات، ثم استدعاء وسيلة Close لإقفال الاتصال. ويجب الحرص على الاحتفاظ بالاتصالات مفتوحة مع مصدر البيانات لأقل وقت ممكن لكى يتم تحرير الموارد التى تستخدمها التطبيقات.

تأهيل فئة بيانات باستخدام موفقات بيانات متعددة

يمكن استخدام أى عدد من الموفقات مقترنا مع فئة بيانات. ويمكن استخدام كل موفق لتعبئة واحد أو أكثر من كائنات جداول فئات البيانات (DataTable) وتسجيل التغييرات فى مصدر البيانات. ويمكن إضافة كائنات العلاقات (DataRelation) وكائنات القيود (Constraint) إلى فئة البيانات محلياً، لتمكيننا من تكوين علاقات بين مصادر متعددة غير متشابهة. على سبيل المثال، يمكن أن تحتوى فئة البيانات على بيانات قاعدة بيانات فى

Microsoft SQL Server، قاعدة بيانات في IBM DB2.

المثال التالى يقوم بتأهيل جدول للعملاء في فئة بيانات باستخدام جدول العملاء في قاعدة بيانات Northwind الموجودة في SQL Server. كما يقوم بتأهيل جدول للأوامر في نفس فئة البيانات باستخدام قاعدة بيانات Northwind الموجودة في Microsoft Access 2000. ويتم ربط جداول فئة البيانات باستخدام كائن علاقات (DataRelation Object)، عرض قائمة العملاء وعرض الأوامر التابعة لكل عميل:

```
Imports System.Data
Imports System.Data.SqlClient
Imports System.Data.OleDb
Public Class Form1
   Inherits System.Windows.Forms.Form
#Region " Windows Form Designer generated code "
   Private Sub Button1_Click (ByVal sender As System.Object, ByVal e
   As System.EventArgs) Handles Button1.Click
      DispRelation ()
   End Sub
   Private Sub DispRelation ()
      Dim custConn As SqlConnection = New SqlConnection ("Data
          Source=localhost;Integrated Security=SSPI;" & _
          "Initial Catalog=northwind;")
      Dim custDA As SqlDataAdapter = New SqlDataAdapter ("SELECT *
          FROM Customers", custConn)

      Dim orderConn As OleDbConnection = New
         OleDbConnection ("Provider=Microsoft.Jet.OLEDB.4.0;" & _
         "Data Source=c:\Program Files\Microsoft Office\" & _
         "Office\Samples\northwind.mdb;")
      Dim orderDA As OleDbDataAdapter = New
         OleDbDataAdapter ("SELECT * FROM Orders", orderConn)

      custConn.Open ()
      orderConn.Open ()
      Dim custDS As DataSet = New DataSet ()
      custDA.Fill (custDS, "Customers")
      orderDA.Fill (custDS, "Orders")
      custConn.Close ()
      orderConn.Close ()
```

```
      Dim custOrderRel As DataRelation =
        custDS.Relations.Add ("CustOrders", _
        custDS.Tables ("Customers").Columns ("CustomerID"), _
        custDS.Tables ("Orders").Columns ("CustomerID"))
      Dim pRow, cRow As DataRow
      For Each pRow In custDS.Tables ("Customers").Rows
        Console.WriteLine (pRow ("CustomerID").ToString ())
        For Each cRow In pRow.GetChildRows (custOrderRel)
          Console.WriteLine (vbTab & cRow ("OrderID").ToString ())
        Next
      Next
    End Sub
End Class
```

التعامل مع البيانات من نوع Decimal فى SQL Server

تقوم فئة البيانات بتخزين البيانات باستخدام أنواع البيانات الموجودة فى نظام NET Framework. وبالنسبة لمعظم التطبيقات، يعتبر ذلك كافيا لتمثيل البيانات الموجودة فى مصدر البيانات. من ناحية أخرى، يمكن أن يسبب هذا التمثيل مشكلة عند استخدام البيانات من نوع Decimal فى SQL Server. ويرجع ذلك إلى أن الحد الأقصى للأرقام الذى يسمح به نظام NET Framework هو ٢٨ رقم على يمين العلامة العشرية، بينما يسمح SQL Server بعدد ٣٨ رقم على يمين العلامة العشرية. فإذا وجد SqlDataAdapter أثناء تنفيذ وسيلة Fill، أن دقة أحد بيانات SQL Server من نوع Decimal تتجاوز ٢٨ رقم، فلن يتم إضافة الصف إلى فئة البيانات وسوف يقع حدث FillError، الذى يمكن استخدامه فى تحديد مقدار الدقة المفقودة والاستجابة بالطريقة المناسبة.

استخدام المعاملات فى موفقات البيانات

يتم تعريف الأوامر المستخدمة فى كائنات أوامر البيانات باستخدام خاصية CommandText. وتشتمل الأوامر التى يجرى تعريفها فى الغالب، على معاملات. فى أثناء التشغيل، تستخدم هذه المعاملات لتمرير قيم إلى عبارات SQL أو الإجراءات المخزنة التى تحتوى عليها كائنات الأوامر. ويمكن تقسيم هذه المعاملات إلى نوعين:

- معاملات اختيار (Selection Parameters). تستخدم فى استخراج مجموعات من البيانات الموجودة فى قاعدة البيانات. وتستخدم هذه المعاملات فى فقرة WHERE

المستخدمة فى عبارة SQL أو إجراء مخزن (Stored Procedure). ويتم الحصول على قيم المعاملات المذكورة فى هذه الفقرة أثناء التشغيل.

معاملات التحديث (Update Parameters). تشمل المعاملات التى تحدد قيم الأعمدة عند تحديث أحد سجلات قاعدة البيانات أو إدخال سجل جديد. وتستخدم هذه المعاملات بالإضافة إلى فقرة Where فى عبارة SQL أو إجراء مخزن.

معاملات الاختيار فى موفقات البيانات

عند اختيار سجلات لتأهيل فئة بيانات، نقوم باستخدام معامل أو أكثر مع فقرة WHERE لكى نستطيع تحديد السجلات المطلوب الحصول عليها أثناء التشغيل. على سبيل المثال، يمكن أن يقوم المستخدم بالبحث فى قاعدة بيانات خاصة بالكتب عن أحد العناوين. لتحقيق ذلك، يمكن استخدام عبارة SQL فى خاصية CommandText بكائن SelectCommand مثل عبارة SQL التالية :

```
SELECT BookId, Title, Author, Price from BOOKS
WHERE (Title LIKE?)
```

أو عبارة SQL التالية:

```
SELECT BookId, Title, Author, Price from BOOKS
WHERE (Title LIKE @title)
```

ويمكن بناء عبارات SQL التى تتطلب معاملات فى Visual Studio باستخدام برنامج Query Builder. وعندما نسحب عناصر من مربع Server Explorer، يمكن أن يقوم Visual Studio بتهيئة المعاملات فى بعض الحالات. وفى هذه الحالة نحتاج إلى استكمال عملية التهيئة يدويا.

معاملات التحديث

تحتوى كائنات InsertCommand، UpdateCommand، DeleteCommand دائما على أوامر بها معاملات، على عكس كائن SelectCommand الذى قد يحتوى على أمر ليس به معاملات. وتحتاج الأوامر الخاصة بكائنات UpdateCommand و InsertCommand إلى معاملات لكل عمود فى قاعدة البيانات نريد تغييره. بالإضافة إلى ذلك، تتطلب العبارات

الخاصة بكائن UpdateCommand وكائن DeleteCommand وجود فقرة WHERE تحتوى على معاملات لكى يتم تحديد السجل المطلوب تحديثه. على سبيل المثال، عند قيام أحد المتسوقين بشراء احتياجاته من الكتب، يمكن استخدام جدول بيانات يمثل عربة الشراء. وسوف يحتوى هذا الجدول على بيانات خاصة بالكتب التى يتم شراؤها، مثل رقم الكتاب، اسم الكتاب، والكمية . عند قيام المستخدم بإضافة أحد الكتب إلى عربة التسوق، يمكن أن يستدعى التطبيق عبارة SQL INSERT لإدراج سجل بالجدول، كما يتضح من الكود التالى:

```
INSERT INTO ShoppingCart
  (BookId, CustId, Quantity)
Values (?, ?, ?)
```

تمثل علامات الاستفهام الثلاثة أماكن المعاملات التى سوف يتم إدخال القيم بها وقت التشغيل. تقابل هذه المعاملات حقل رقم الكتاب، حقل اسم الكتاب، وحقل الكمية على الترتيب. ويمكن استخدام معاملات ذات أسماء، كما يتضح من الكود التالى بيانه:

```
INSERT INTO ShoppingCart
  (BookId, CustId, Quantity)
Values (@bookid, @custid, @quantity)
```

وعندما يقرر المشترى تغيير شيئا ما عن أحد البنود فى عربة التسوق، مثل تغيير الكمية، يمكن أن يستدعى التطبيق عبارة SQL UPDATE لتغيير البند فى جدول البيانات باستخدام الصيغة التالية:

```
UPDATE ShoppingCart
  SET (BookId = ?, CustId = ?, Quantity = ?)
WHERE (BookId = ? AND CustId = ?)
```

وعند استخدام المعاملات ذات الأسماء، تكون الصيغة كما يلى:

```
UPDATE ShoppingCart
  SET (BookId = @bookid, CustId = @custid, Quantity = @quantity)
WHERE (BookId = @bookid AND CustId = @custid)
```

فى هذه العبارة، يتم تعبئة المعاملات فى فقرة Set بالقيم الحديثة الخاصة بالسجل المستهدف تغييره. بينما تقوم فقرة WHERE بتحديد السجل المطلوب تحديثه ويتم تأهيل هذه المعاملات بالقيم الأصلية الموجودة فى السجل. ويمكن للمستخدم أيضا القيام بحذف

أحد البنود من عربة التسوق. وفى هذه الحالة، يقوم التطبيق باستدعاء عبارة DELETE التى تأخذ الصيغة التالية عند استخدام المعاملات بالأماكن:

```
DELETE FROM ShoppingCart
WHERE (BookId = ? AND CustId = ?)
```

أو تأخذ الصيغة التالية، بالنسبة للمعاملات ذات الأسماء:

```
DELETE FROM ShoppingCart
WHERE (BookId = @bookid AND CustId = @custid)
```

العلاقة بين مجموعة المعاملات وبين كائنات المعاملات

للسماح بتمرير قيم المعاملات فى وقت التشغيل، يحتوى كل كائن من كائنات الأوامر الأربعة الموجودة فى موفق البيانات على خاصية Parameters. تحتوى هذه الخاصية على مجموعة من كائنات المعاملات المنفردة التى تقابل أماكن حجز المعاملات فى عبارة. بالنسبة للمعاملات الخاصة بكائن OleDBDataAdapter، تكون مجموعة المعاملات من نوع OleDBParameterCollection، وتكون المعاملات المفردة من نوع OleDBParameter. وبالمثل تكون معاملات الأمر فى كائن SQLDataAdapter من نوع SqlParameter فى مجموعة من نوع SqlParameterCollection.

عند استخدام Data Adapter Configuration Wizard لإعداد موفق البيانات، يتم ضبط وتهيئة مجموعة المعاملات أتوماتيكيا بالنسبة لأوامر موفق البيانات الأربعة. وعند القيام بسحب عناصر من Server Explorer إلى نموذج أو مكون، يقوم Visual Studio بتنفيذ عمليات الإعداد التالية:

- عند سحب جدول أو بعض الأعمدة، يقوم Visual Studio بتكوين كائن SelectCommand (على وجه الخصوص عبارة SQL SELECT) بدون معاملات، وتكوين كائنات UpdateCommand، كائنات InsertCommand، وكائنات DeleteCommand ذات المعاملات. وعند الرغبة فى ربط كائن SelectCommand بمعاملات، يجب القيام بذلك يدويا.
- إذا سحبنا إجراء مخزن إلى المصمم، يقوم Visual Studio بإنتاج كائن SelectCommand به المعاملات المطلوبة بواسطة الإجراء المخزن. من ناحية أخرى، عندما نريد كائنات

UpdateCommand، InsertCommand، DeleteCommand، يجب إعدادها مع معاملاتها بأنفسنا.

وبصفة عامة عندما نرغب فى تكوين استعلامات ترتبط بمعاملات فى موفق البيانات، يجب استخدام أداة Data Adapter Configuration Wizard للقيام بهذه المهمة.

هيكل مجموعة المعاملات

البنود الموجودة فى مجموعة المعاملات (Parameter Collection) بأحد كائنات الأوامر تقابل المعاملات المطلوبة بواسطة كائن الأمر. إذا كان كائن الأمر يحتوى على عبارة SQL، فإن بنود المجموعة تقابل المواقع المحجوزة للمعاملات فى العبارة. على سبيل المثال، مجموعة UDATE التالية، تتطلب مجموعة من خمسة معاملات:

```
UPDATE ShoppingCart
  SET (BookId = ?, CustId = ?, Quantity = ?)
WHERE (BookId = ? AND CustId = ?)
```

وفيما يلى نفس العبارة مع استخدام المعاملات ذات الأسماء:

```
UPDATE ShoppingCart
  SET (BookId = @bookid, CustId = @custid, Quantity = @quantity)
WHERE (BookId = @bookid AND CustId = @custid)
```

وإذا كان كائن الأمر يحتوى على إجراء مخزن، فإن عدد المعاملات فى المجموعة يتم تقريرها بواسطة الإجراء. ويمكن ألا تقابل معاملات المجموعة بالضبط، المواقع الموجودة فى عبارة SQL التى يحتوى عليها الإجراء. ويمكن تسمية المعاملات فى الإجراء المخزن أيضا. وفى هذه الحالة لا يكون موقع المعامل فى المجموعة مهما، ويجرى استخدام خاصية ParameterName فى كل بند بالمجموعة لمناظرته مع المعامل الموجود فى الإجراء المخزن.

وعند القيام بتهيئة المعاملات يدويا، يجب أن نحدد بالضبط المعاملات التى يتطلبها الإجراء المخزن. كما يجب الأخذ فى الاعتبار أن هناك الكثير من الإجراءات المخزنة التى تعيد قيمة، مما يترتب علية تمرير القيمة من مجموعة المعاملات إلى التطبيق. وقد يحتوى الإجراء المخزن على العديد من عبارات SQL، ولذلك يجب التحقق من أن مجموعة المعاملات (Parameters Collection) تحتوى على كل القيم المطلوبة.

وعندما تكون المعاملات بدون أسماء فى الأوامر التى تنفذ بقاعدة البيانات، يجب

مناظرة بنود المجموعة مع المعاملات المطلوبة على أساس المكان. وعندما يحتوى الأمر على إجراء مخزن يعيد قيمة، يجب تخصيص البند الأول فى المجموعة، وهو البند رقم صفر، للقيمة المرتجعة من الأمر إلى التطبيق.

على هذا الأساس، يمكن الإشارة إلى كائنات المعاملات الفردية باستخدام مكانها فى فهرس المجموعة. كما يمكن الإشارة إليها بالاسم عن طريق استخدام خاصية ParameterName. على سبيل المثال، العبارتان التاليتان متساويتان، بافتراض أن المعامل الثانى فى المجموعة يسمى Tite_Keyword:

```
OleDbDataAdapter1.SelectCommand.Parameters (1).Value = titleKeyword
OleDbDataAdapter1.SelectCommand.Parameters ("Title_Keyword").Value =
titleKeyword
```

وبصفة عامة، يعتبر استخدام المعاملات أفضل كثيرا من الإشارة إلى المعاملات عن طريق موقعها فى فهرس. لأن ذلك يقلل من الحاجة إلى إجراءات الصيانة عند تغيير عدد المعاملات ويرفع عن عاتقنا عبء تذكر أن الإجراء المخزن يعيد إلينا قيمة. ومع أن استخدام أسماء للمعاملات يترتب علية بعض الأعباء الإضافية، إلا أن ذلك يمكن تعويضه بسهولة البرمجة وسهولة الصيانة.

بناء قيم المعاملات

هناك طريقتان لتحديد قيم المعاملات:

- ضبط خاصية Value بالمعامل بطريقة صحيحة.
- مناظرة المعاملات مع الأعمدة فى جدول فئة بيانات، لكى يتم استخراج القيم من صفوف البيانات عندما نحتاج إلى ذلك.

ونحتاج إلى الضبط الصريح لقيم معاملات الاختيار (Selection Parameters) عند تأهيل فئة بيانات أو استدعاء أحد الأوامر (Calling Command). على سبيل المثال، يمكن أن يحتوى تطبيق على مربع نص لكى يستطيع المستخدم إدخال الاسم الأخير للمؤلف بغرض البحث عنة، ثم ضبط قيمة المعامل على القيمة التى يتم إدخالها بمربع النص قبل استخدام وسيلة Fill بموفق البيانات. وفيما يلى الكود اللازم لتطبيق ذلك:

```
Dim dsAuthors1 As DataSet = New DataSet ()
```

```
Dim lastName As String
Dim pRow As DataRow
Dim myConnection As SqlConnection = New SqlConnection ("data source=localhost; integrated security=SSPI; Initial Catalog=pubs;")
Dim SqlDataAdapter1 As SqlDataAdapter = New SqlDataAdapter ("Select * from Authors", myConnection)
SqlDataAdapter1.SelectCommand.Parameters.Add ("@Last_Name", sqlDbType.NChar, 15, "au_lname")
lastName = txtLastName.Text
SqlDataAdapter1.SelectCommand.Parameters ("@Last_Name").Value = lastName
myConnection.Open ()
SqlDataAdapter1.Fill (dsAuthors1, "Authors")
For Each pRow In dsAuthors1.Tables ("Authors").Rows
    Console.WriteLine (pRow ("au_fname").ToString ())
Next
myConnection.Close ()
```

ويتم استخدام قيم المعاملات المتناظرة أثناء عمليات التحديث. حيث تقوم وسيلة Update عند استدعائها فى موفق البيانات بتتبع السجلات فى جدول فئة بيانات والقيام بعملية التحديث المناسبة لكل سجل. فى هذه الحالة، تكون قيم المعاملات متاحة فى صورة أعمدة بسجلات فئة البيانات. على سبيل المثال، عندما تصل عملية التحديث إلى سجل جديد فى جدول فئة بيانات يستلزم استدعاء عبارة INSERT فى قاعدة بيانات، يمكن قراءة قيم فقرة Value فى عبارة INSERT مباشرة من السجل.

ومن الممكن ضبط معاملات التحديث صراحة. لأن موفق البيانات يدعم حدث RowUpdating الذى يجرى استدعاؤه فى كل مرة يتم فيها تحديث أحد الصفوف. ويمكن إنشاء إجراء معالجة لهذا الحدث وضبط قيم المعاملات. مما يؤدى إلى توفير رقابة محكمة على قيم المعاملات، ويسمح لنا بتنفيذ عمليات مثل إنشاء قيم المعاملات ديناميكيا قبل كتابتها فى سجل قاعدة بيانات.

إعداد المعاملات فى موفقات البيانات

تقوم كائنات أوامر موفق البيانات التى تشمل كائن InsertCommand، كائن SelectCommand، كائن UpdateCommand، وكائن DeleteCommand باستخدام عبارات SQL أو الإجراءات المخزنة التى تتطلب استخدام المعاملات. ويتم تهيئة هذه المعاملات أتوماتيكيا عند استخدام مربع حوار Data Adapter Configuration Wizard فى تكوين

موفقات البيانات. من ناحية ثانية، يمكن تهيئة هذه المعاملات يدويا أو إجراء بعض التغييرات فى مجموعات المعاملات. لتهيئة معاملات موفق بيانات:

١. نقوم بتكوين موفق بيانات.

٢. فى مصمم النماذج، نختار موفق البيانات ثم نفتح نافذة Properties.

٣. نقوم بتوسيع كائن Command الذى نريد تهيئة معاملاته. فى خاصية Parameters، ننقر نقاط الحذف لفتح مربع حوار Parameter Collection Editor.

٤. لإنشاء كائن Parameter جديد، ننقر Add.

٥. نضع المعامل الجديد فى الموقع الصحيح على الفهرس فى مجموعة هذا المعامل بالنقر على الاسم الموجود أسفل Sort لتحريكها.

٦. لضبط خصائص معامل، نختار ذلك المعامل فى قائمة Members ثم نستخدم شبكة Property على الجانب الأيمن. الجدول رقم (٢٢) يعرض الخصائص التى يجب ضبطها.

الخاصية	البيان
SourceColumn	اسم عمود فى فئة بيانات يتم قراءة قيمة المعامل منة. تستخدم هذه الخاصية مع المعاملات التى تملئ فقرة Values فى عبارات INSERT، UPDATE، أو DELETE، أو الإجراءات المخزنة المقابلة.
SourceVersion	تحدد الإصدار الخاص بسجل البيانات الذى يجب استخدامه عند استخراج قيمة المعامل باستخدام خاصية SourceColumn.
Value	قيمة صريحة يتم تحديدها للمعامل. ويتم ضبط هذه الخاصية غالبا فى وقت التشغيل ولها الأولوية على خاصية SourceColumn.

الخاصية	البيان
NamedParameter	قيمة منطقية تشير إلى أن المعامل يمثله متغير أو موقع
DBType, Precision, Scale, Size	معلومات عن نوع البيانات فى مخزن البيانات الخاص بقيمة المعامل.
Direction	تحديد ما إذا كانت قيمة المعامل يجرى تمريرها إلى الأمر أو فى الاتجاه المعاكس.
ParameterName	اسم يستخدم للإشارة إلى المعامل فى مجموعة المعاملات بدلا من استخدام موقعة فى الفهرس.

جدول ٢٢

٧. ننقر على زر Ok لإقفال نافذة Parameter Collection Editor.

٨. نكرر الخطوات من ٣ إلى ٧ لتهيئة المعاملات لكائنات الأوامر الأخرى.

المناظرة بين البيانات

عندما يقوم موفق بيانات بقراءة البيانات من أحد المصادر، يستخدم المناظرة (Mapping) فى تقرير أين سيضع البيانات فى الجدول المقابل بفئة البيانات. تقوم عملية المناظرة بربط أسماء الأعمدة فى مصدر بيانات مع الأعمدة المقابلة فى جدول بفئة البيانات. على سبيل المثال، المعلومات الموجودة بحقل au_id فى مصدر بيانات، يمكن أن تخص عمود يدعى author_id_number فى جدول فئة بيانات. وعند استخدام أدوات Visual Studio لتكوين فئة بيانات من المعلومات الموجودة فى مصدر بيانات، يكون الوضع الإفتراضى هو استخدام أسماء الأعمدة فى مصدر البيانات لتسمية تلك الموجودة فى جدول فئة البيانات. ومع ذلك، هناك بعض الحالات التى لا تتطابق فيها الأسماء الموجودة فى مصدر البيانات مع تلك الموجودة فى فئة البيانات:

- عند تكوين فئة بيانات من مخطط (Schema) قائم يستخدم أسماء مختلفة.
- تغيير أسماء عناصر البيانات في فئة البيانات لكى تكون أكثر دلالة.
- عندما نريد التحكم في أسماء أعضاء البيانات النوعية عند إنتاج فئة بيانات بواسطة موفق بيانات.

هيكل البيانات المستخدم في المناظرة بين جدول مصدر البيانات وجدول فئة البيانات

تتم المناظرة بين الجداول باستخدام كائنات DataTableMapping، التى توجد فى خاصية TableMappings الموجودة فى موفق البيانات. يحتوى كائن DataTableMapping على مجموعة تتكون من جدول فى مصدر البيانات والجدول المقابل فى فئة البيانات. ومن المعتاد وجود واحد فقط من هذه البنود لأن موفقات البيانات فى الوضع الإفتراضى، ترتبط مع جدول واحد فى مصدر البيانات وجدول واحد فى فئة البيانات. ومع ذلك، يمكن أن ينتج عن الإجراء المخزن العديد من مجموعات الصفوف. فى هذه الحالة، يتم مناظرة الجدول الثانى والجداول التى تأتى بعدة باستخدام كائن المناظرة الثانى والكائنات التى تأتى من بعدة فى مجموعة TableMappings. ويحتوى كل كائن مناظرة على خصائص لتحديد جدول مصدر البيانات، جدول فئة البيانات، وكائنات المناظرة بين الأعمدة (ColumnMappings) التى يتكون منها الجدولان.

تنفيذ مناظرة الجداول

عند استدعاء وسيلة Fill فى موفق بيانات، يقوم موفق البيانات بالعمليات التالية لمعرفة مكان الكتابة فى فئة البيانات:

١. يستطلع موفق البيانات (Data Adapter) اسم عمود مصدر بيانات فى كائن TableMappings.

٢. عـند العثور على اسم عمود المصدر المستهدف، يتم الحصول على اسم العمود المقابل فى فئة البيانات.

٣. باستخدام الاسم الذى يتم الحصول علية فى الخطوة الثانية، يقوم الموفق بكتابة

بيانات عمود المصدر فى العمود المقابل بفئة البيانات.

وهناك بعض الظروف التى قد تمنع موفق البيانات من تنفيذ ترتيب العمليات المحددة أعلاه. أهم هذه الظروف هى:

- لم يتم تعريف عمود مناظر لعمود مصدر البيانات فى خاصية TableMappings.
- لا يوجد فعليا عمود فى فئة البيانات يناظر عمود مصدر البيانات.

وهذه الحالات ليست بالضرورة أخطاء، لأن الموفق يمكن أن يستمر فى تعبئة فئة البيانات فى ظل حدوث هذه الظروف. ويدعم موفق البيانات اثنين من الخصائص التى تسمح لنا بتحديد ما يمكن أن يحدث عند حدوث أى من هذين الظرفين. الخاصية الأولى هى خاصية MissingMappingAction التى تقوم بتحديد الفعل الذى يجب أن يقوم به الموفق عند فقد التناظر. ويمكن أن تحتوى هذه الخاصية على القيم التالية:

- قيمة Passthrough التى تأمر موفق البيانات بالبحث عن عمود فى فئة البيانات بنفس الاسم، ويعتمد السلوك المتبع فى هذه الحالة على القيمة الموجودة فى خاصية MissingSchemaAction.
- القيمة الثانية هى Ignore، التى تعنى عدم تحميل البيانات بأعمدة فئة البيانات التى لا يتم مناظرتها بطريقة صحيحة.
- والقيمة الأخيرة هى Error، التى يترتب عليها رفع رسالة خطأ.

وتسمح لنا خاصية MissingSchemaAction بتحديد ما يجب أن يحدث عندما يحاول الموفق كتابة بيانات فى عمود غير معرف فى مخطط فئة البيانات. القيم الممكنة لهذة الخاصية تشمل:

- القيمة الأولى هى Add، التى يترتب عليها إضافة الجدول أو العمود إلى فئة البيانات.
- القيمة الثانية هى AddWithKey، التى تعنى إضافة جدول البيانات أو العمود إلى فئة البيانات والى المخطط مرفقا مع بيانات عن المفتاح الأساسي (Primary Key).
- القيمة الثالثة Ignore، التى عدم إضافة الجدول أو العمود غير الممثل فى مخطط

فئة البيانات.

- القيمة الأخيرة هى Error، التى تعنى قيام موفق البيانات برفع رسالة خطأ.

وبصفة عامة، نقوم بضبط كل من الخاصتين معا لتوفير الاحتياجات الخاصة للتطبيق. ضبط الخاصية الأولى على Passthrough، وضبط الخاصية الثانية على القيمة Add، له نفس تأثير النسخ التلقائي لاسم الجدول أو العمود من المصدر إلى فئة البيانات. وعلى العكس من ذلك، يمكن الضبط على القيمة Error فى التطبيقات التى يكون مخطط فئة البيانات بها محدد بدقة. فى هذه الحالة، يمكن أن يمثل الحصول على البيانات من المصدر بدون تعريف عمود مستهدف بوضوح فى فئة البيانات، مخالفة لقاعدة عمل أو خطأ من نوع أخر. ويتم الضبط على قيمة Ignore عندما نريد أن يكون هناك تحقق من أن البيانات المحملة فى فئة البيانات هى فقط التى تم تعريفها صراحة فى المخطط ومناظرتها فى خاصية TableMappings. ويعتبر هذا مفيدا عندما يقوم الموفق باستدعاء إجراء مخزن أو عبارة SQL تؤدى إلى الحصول على أعمدة أكثر مما نحتاجه فى فئة البيانات.

مناظرة أعمدة جدول مصدر بيانات مع أعمدة جدول فئة بيانات

تسمح لنا عملية المناظرة فى موفق البيانات ببناء مقابلة بين الأعمدة الموجودة فى مصدر البيانات وبين الأعمدة الموجودة فى فئة بيانات. لتنفيذ هذه المناظرة:

١. نقوم بإنشاء موفق بيانات (Data Adapter).

٢. فى مصمم النموذج، نختار موفق البيانات ونفتح نافذة Properties.

٣. بجانب خاصية TableMappings، ننقر نقاط الحذف. يؤدى ذلك إلى عرض مربع حوار Table Mappings، المبين بالشكل رقم (١٤).

٤. عندما نريد مناظرة أسماء أعمدة من مصدر البيانات مع أسماء أعمدة فى فئة بيانات قائمة، نضع علامة الاختيار أمام Use a Dataset to suggest table and column names، ثم نختار فئة البيانات من القائمة المنسدلة مقابل dataset. يترتب على ذلك عرض الأعمدة الخاصة بالجدول الأول بفئة البيانات على الجانب الأيمن فى شبكة Column Mappings.

٥. تحت Source Table، نختار جدول مصدر البيانات الذى نريد مناظرته. فإذا كانت فئة البيانات تحتوى على جدول واحد، فسوف نجد القيمة الافتراضية Table.

٦. تحت Dataset Table، نختار الجدول المناظر فى فئة البيانات.

٧. نقوم بتعديل عمليات التناظر المطلوبة التى من الممكن أن تشمل ما يلى:

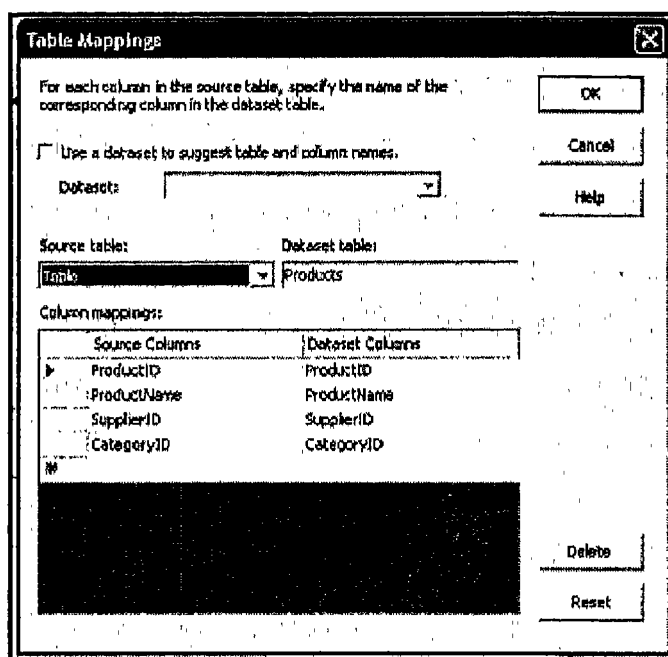
أ. اختيار أعمدة مختلفة فى أى من قائمة أعمدة مصدر أو فئة البيانات لتحديد العمود الذى يناظر الأخر.

ب. حذف أعمدة من عملية المناظرة. ونقوم بذلك عندما يعيد إلينا الموفق أعمدة أكثر مما نحتاج إلية.

ت. إضافة أعمدة. ويمكن القيام بذلك عندما نعلم أن هناك أعمدة سوف يتم إضافتها فى وقت التشغيل غير ممثلة فى وقت التصميم، وأيضا عند حذف أحد الأعمدة ثم الحاجة إلية مرة أخرى.

٨. نقوم بالنقر على Ok بعد الانتهاء من عملية المناظرة.

٩. وعندما نريد معالجة الأخطاء المحتملة، نقوم بضبط خاصية MissingMappingAction وخاصية MissingSchemaAction فى نافذة Properties.



شكل رقم ١٤

المشاهدة المبدئية لنتائج موفق البيانات

بعد الانتهاء من تهيئة موفق بيانات، يمكن اختبار كيفية قيام الموفق بتأهيل فئة البيانات باتباع الخطوات التالية:

١. نقوم بإنشاء موفق بيانات.

٢. ننقر بزر الماوس الأيمن على موفق البيانات فى النموذج.

٣. من القائمة المختصرة، نختار Preview. يترتب على ذلك فتح نافذة Data Preview

٤. نختار موفق بيانات من قائمة Data Adapter، أو نختار All Data Adapters.

٥. إذا كان الأمر المشار إلية فى خاصية SelectCommand يتطلب معاملات، نقوم بإدخال قيم اختبار للمعاملات فى شبكة Parameters.

٦. ننقر Fill Dataset لتنفيذ الأمر.

٧. نقوم بتكرار نفس الخطوات مع موفقات البيانات الأخرى أو استخدام قيم مختلفة

للمعاملات. يترتب على ذلك إلحاق عرض كل مجموعة بيانات ناتجة عن المشاهدة المبدئية بمجموعة المشاهدة الأخرى الموجودة بالفعل فى Dataset Preview. ويمكن البدء مع فئة بيانات خالية بالنقر على Clear Table.

العمليات المباشرة مع مصادر البيانات

فى كثير من الحالات، نقوم بتصميم استراتيجية الوصول إلى البيانات على أساس استخدام موفقات البيانات (Data Adapters) التى تقوم بتعبئة فئة بيانات (Datasets) ونقل تغييرات فئات البيانات إلى مصدر البيانات. وفى حالات أخرى، يمكن أن يكون الاتصال المباشر مع قاعدة البيانات والعمل معها أكثر فائدة من استخدام فئات البيانات والموفقات. تشمل حالات تفضيل استخدام نموذج الوصول المباشر إلى البيانات:

- تنفيذ الاستعلامات على البيانات المطلوب قراءتها فقط.
- تنفيذ الاستعلامات التى تعيد قيمة منفردة، مثل عملية حسابية أو نتيجة دالة تجميع.
- إنشاء وتعديل هياكل قواعد البيانات، مثل الجداول والإجراءات المخزنة.
- التنفيذ الديناميكي لأوامر SQL الخاصة بتحديث البيانات، إدراج السجلات، وحذف السجلات مباشرة فى قاعدة البيانات بدلا من تحديث فئة البيانات ثم نسخ التغييرات إلى قاعدة البيانات.
- تنفيذ الأوامر التى تعيد بيانات فى صيغة XML من SQL Server الإصدار ٧ وما بعدة.

إن تقنية ADO.NET تفترض أن نموذج الوصول إلى البيانات يتكون من فتح اتصال مع مصدر البيانات، الحصول على البيانات أو تنفيذ عملية، ثم إقفال الاتصال. ويوفر ADO.NET طريقتين للعمل مع هذا النموذج. الطريقة الأولى هى تخزين البيانات فى فئة بيانات (dataset)، التى تعتبر ذاكرة وسيطة لتخزين السجلات التى يمكن العمل معها أثناء عدم الاتصال مع مصدر البيانات، والطريقة الثانية هى تنفيذ العمليات مباشرة على قاعدة البيانات. لاستخدام فئة البيانات، نقوم بإنشاء مثل من تصنيف فئة البيانات (

DataSet Class) ثم استخدام موفق البيانات (Data Adapter) لتعبئتها بالبيانات من المصدر. يجرى بعد ذلك العمل مع البيانات فى فئة البيانات ثم تحديث مصدر البيانات بالتغييرات التى تحدث فى تلك البيانات. وفى الطريقة الثانية، نستخدم كائن أمر بيانات (Data Command Object) يشتمل على مرجع إلى عبارة SQL أو إلى إجراء مخزن (Stored Procedure). ثم نقوم بفتح اتصال مع قاعدة البيانات، تنفيذ الأمر، وأخيرا إقفال الاتصال. فإذا كانت نتيجة تنفيذ الأمر هى الحصول على مجموعة من السجلات، نستخرج البيانات منها باستخدام كائن قارئ بيانات (DataReader Object).

كل طريقة من هاتين الطريقتين لها مزايا محددة، ويجب اختيار الطريقة التى تتناسب مع متطلبات الوصول إلى البيانات. ويمكن حصر المزايا الناتجة عن استخدام نموذج تخزين البيانات فى فئة واستخدام كائن موفق بيانات لقراءة وكتابة البيانات فى قاعدة البيانات فيما يلى:

١. العمل مع جداول متعددة لأن فئة البيانات يمكن أن تحتوى على عدد من الجداول التى تحتفظ بها فى صورة كائنات منفصلة. ويمكن العمل مع هذه الجداول كل على انفراد أو الحركة بينها فى صورة جداول أصلية وجداول تابعة.

٢. معالجة البيانات الواردة من مصادر متعددة. يمكن أن تمثل الجداول فى فئة البيانات مصادر بيانات مختلفة. وبمجرد أن تصبح هذه الجداول فى فئة البيانات، يمكن معالجتها وإنشاء العلاقات بينها فى شكل متجانس كما لو أنها قادمة من مصدر واحد.

٣. تحريك البيانات بين الطبقات فى التطبيقات التى تحتوى على أكثر من طبقة. حيث يمكن تحريك البيانات بين طبقة العرض، طبقة الضوابط التجارية، وطبقة البيانات.

٤. تبادل البيانات مع التطبيقات الأخرى. توفر فئة البيانات طريقة قوية لتبادل البيانات مع المكونات الأخرى فى التطبيق ومع التطبيقات الأخرى. وتشتمل فئات البيانات على دعم مكثف لخواص مثل تسجيل البيانات فى صورة XML وقراءة وكتابة مخططات XML.

٥. ربط البيانات مع أدوات التحكم عند العمل مع النماذج. حيث أنة من الأسهل ربط أدوات التحكم مع البيانات فى فئة بيانات أكثر من استخدام الكود لتحميل البيانات فى أدوات التحكم بعد تنفيذ الأمر.

٦. الاحتفاظ بالسجلات لإعادة استخدامها. حيث تسمح لنا فئة البيانات بالعمل مع نفس السجلات بالتكرار بدون إعادة طلبها من قاعدة البيانات. كما يمكن ترشيح وفرز البيانات بسهولة.

٧. سهولة البرمجة. عند العمل مع فئة بيانات، يمكن إنتاج ملف تصنيف يمثل هيكل بنائها فى صورة كائنات. على سبيل المثال،جدول العملاء (Customer Table) فى فئة البيانات يمكن الوصول إلية على أنة كائن (Dataset.Customer Object). يؤدى ذلك إلى تسهيل عمليات البرمجة.

ويترتب على تنفيذ عمليات قاعدة البيانات مباشرة، المزايا التالية:

١. القيام بوظائف إضافية. حيث توجد بعض العمليات، مثل عمليات توصيف هياكل البيانات (DDL)، لايمكن تنفيذها إلا عن طريق تنفيذ أوامر البيانات (Data Commands).

٢. تحكم أكثر فى التنفيذ. يترتب على تنفيذ أوامر البيانات مباشرة الحصول على تحكم أكثر فى عملية ووقت تنفيذ عبارات SQL أو الإجراءات المخزنة.

٣. تحمل أعباء أقل. يمكن عن طريق القراءة والكتابة المباشرة فى قاعدة البيانات، تجاوز تخزين البيانات فى فئة البيانات. وبالنظر إلى أن فئة البيانات تتطلب ذاكرة، .فإن عدم استخدامها يجنب بعض الأعباء.

٤. فى بعض الظروف يمكن أن يترتب على التنفيذ المباشر للأوامر برمجة أقل. على سبيل المثال، يترتب على استخدام قارئ البيانات تجنب الخطوات الإضافية الخاصة بإدارة فئة البيانات.

العمل مع أوامر البيانات

يحتوى أمر البيانات على مرجع إلى عبارة SQL أو إجراء مخزن يمكن تنفيذه مباشرة.

وأمر المبيعات هو مثل من تصنيف OleDbCommand أو تصنيف SqlCommand. ويستخدم أمر OleDbCommand مع أى مورد OLE-DB، بينما يستخدم أمر SqlCommand مع SQL Server الإصدار رقم ٧ والإصدارات التى بعدة. ويحتوى كائن أمر البيانات على مجموعة من الخصائص التى توفر جميع المعلومات الضرورية لتنفيذ أمر على قاعدة بيانات. تشمل هذه المعلومات :

- اتصال (Connection) يستخدمه الأمر للوصول إلى قاعدة البيانات.
- نص (Text) يحتوى على أمر SQL أو اسم إجراء مخزن.
- معاملات (Parameters) لأن الأمر قد يتطلب تمرير قيم إلية فى صورة معاملات داخلة أو ينتج عنة قيم توضع فى معاملات خارجة.

ويستخدم لتنفيذ أمر البيانات وسيلة مناسبة للنتائج المتوقع الحصول عليها. على سبيل المثال، إذا كنا نتوقع الحصول على مجموعة سجلات، نستدعى وسيلة ExecuteReader التى تعيد سجلات إلى قارئ البيانات. وعند تنفيذ أمر تحديث، إدراج، أو حذف، نستدعى وسيلة ExecuteNonQuery، التى ينتج عنها قيمة تشير إلى عدد السجلات المتأثرة بتلك العمليات.

الحصول على مجموعات سجلات متعددة

يتمثل الاستعمال النموذجي لأوامر البيانات فى الحصول على مجموعة سجلات. ويمكن الحصول على مجموعات السجلات بأكثر من طريقة. أحد هذه الطرق، أن الأمر يستخدم إجراء مخزن ينتج عن تنفيذه مجموعات سجلات متعددة. والطريقة الثانية هى أن الأمر يحتوى على اثنين أو أكثر من عبارات SQL أو الإجراءات المخزنة. فى هذه الحالة، يتم تشغيل العبارات أو الإجراءات منفصلة عن بعضها وينتج عنها مجموعات متعددة من السجلات فى استدعاء واحد. ويترتب على الحصول على مجموعات متعددة من السجلات، الاستخدام الأمثل لاتصال واحد مفتوح. ومن الاستخدامات النموذجية فى هذه الحالة:

- تشغيل عدة استعلامات، كل واحد منها ينتج عنة مجموعة سجلات.
- تشـغيل عـبارة UPDATE أو عبارة INSERT متبوعة بعبارة SELECT التى ينتج عنها

نسـخة حديـثة مـن السجلات المعدلة، بما فيها القيم التى تحددها قواعد البيانات مثل القيم المتزايدة.

وعندما نقوم بتحديد عبارات أو إجراءات متعددة للأمر، يجب أن يكونوا جميعا من نفس النوع. على سبيل المثال، يمكن استخدام عبارات SQL متتابعة أو إجراءات مخزنة متتابعة. ولا يمكن مزج عمليات استدعاء إجراءات مخزنة مع عمليات استدعاء عبارات SQL فى أمر واحد.

الأوامر المستخدمة فى موفقات البيانات

تستخدم موفقات البيانات الأوامر للقراءة والكتابة فى قاعدة بيانات. ويمكن أن تحتوى موفقات البيانات على أربعة من كائنات الأوامر، على أساس أمر لكل خاصية من خواص الأمر وهى: خاصية UpdateCommand، خاصية SelectCommand، خاصية InsertCommand، وخاصية DeleteCommand. ويقوم موفق البيانات بتنفيذ الأوامر التى يجب علينا تنفيذها بطريقة فعالة. على سبيل المثال، عند استدعاء وسيلة Fill، يقوم أمر البيانات بتنفيذ أمر البيانات الموجود فى خاصية SelectCommand، ويستخدم كائن DataReader لتعبئة مجموعة السجلات فى جدول فئة البيانات الذى يتم تحديده. وبالمثل، عند استدعاء وسيلة Update، يقوم موفق البيانات بتنفيذ الأمر المناسب فى خاصية InsertCommand، UpdateCommand، DeleteCommand بالنسبة لكل سجل يتم تغييره فى فئة البيانات. ويتحقق موفق البيانات أيضا من تمرير المعاملات المناسبة مع تلك الأوامر.

ويمكن أن تقوم أوامر البيانات بتنفيذ هذه المهام التى يقوم بها موفق البيانات بالإضافة إلى توفير قدر أكبر من التحكم فى كيفية ووقت التنفيذ.

تكوين أوامر البيانات

لتكوين أوامر البيانات فى Visual Studio، نقوم بتنفيذ عدد من الخطوات، بعضها فى وقت التصميم والأخرى فى وقت التشغيل، وفى الوضع النموذجي يتم ذلك باستخدام الكود.

فى وقت التصميم، ننفذ الخطوات التالية:

- نضيف أمر بيانات إلى النموذج.
- نحدد أمر SQL أو الإجراء المخزن.
- نضبط المعاملات المطلوبة.

فى وقت التشغيل، ننفذ الخطوات التالية:

- نضبط قيم المعاملات على نريد تمريرها إلى أمر البيانات. وفى الغالب، نحصل على قيم هذه المعاملات من المعلومات التى يدخلها المستخدم فى النموذج أو التى يجرى تمريرها من مكون إلى أخر.
- ننفذ أمر البيانات، وبالتالى تشغيل عبارة SQL أو الإجراء المخزن الذى يحتوى علية.
- نقرأ القيمة أو القيم الناتجة من أمر البيانات. وتختلف الطريقة المستخدمة فى قراءة النتائج بناءا على نوع الأمر الذى ننفذه وعلى النتيجة العائدة منة.

إضافة أوامر البيانات إلى النماذج

قبل استخدام أمر بيانات فى نموذج، يجب تنفيذ الخطوات التالية:

- نضيف كائن اتصال إلى النموذج، إذا لم يكن هناك واحدا منها.
- من صفحة ملصق Data فى مربع ToolBox، نسحب كائن OleDbCommand أو كائن SqlCommand إلى النموذج.
- نضبط خصائص كائن الأمر طبقا للجدول رقم (٢٣):

الخاصية	الإيضاح
(Name)	الاسم المستخدم للإشارة إلى الأمر
Connection	مرجع إلى كائن الاتصال الذى سوف يستخدمه الأمر
CommandType	قيمة تشير إلى أن الأمر عبارة SQL أو إجراء مخزن
CommandText	الأمر الذى سوف يجرى تنفيذه، سواء كان عبارة SQL أو إجراء مخزن
Parameters	مجموعة من الكائنات المستخدمة لتمرير القيم من والى الأمر

جدول ٢٣

معاملات أمر البيانات

عند تنفيذ أوامر البيانات مباشرة على قاعدة بيانات، فإن عبارات SQL أو الإجراءات المخزنة تتطلب فى الغالب استخدام المعاملات. على سبيل المثال، تتطلب عبارة Update معاملات مثل الموضحة بالكود التالى:

```
UPDATE Employees
SET LastName = @LastName, FirstName = @FirstName, BirthDate = @BirthDate
WHERE (EmployeeID = @Emp_id)
```

عند تنفيذ هذه العبارة، يجب تزويد القيم الخاصة بكل المعاملات المذكورة. وللقيام بذلك،نستخدم كائنات Parameter. تدعم أوامر البيانات مجموعة Parameters التى تحتوى على مجموعة من الكائنات من نوع تصنيف OleDbParameter أو SqlParameter. ويوجد كائن Parameter فى المجموعة لكل معامل نحتاج إلى تمريرة. بالإضافة إلى ذلك، إذا كنا نستدعى إجراء مخزن، قد نحتاج إلى معامل إضافي لاستقبال القيمة الناتجة عن العملية. وقبل تنفيذ أحد الأوامر، يجب تحديد قيمة لكل معامل فى ذلك الأمر. ولتحديد قيمة معامل فى مجموعة معاملات الأمر، نضبط خاصية Value على القيمة التى نمررها. المثال التالى، يوضح كيفية ضبط المعاملات قبل تنفيذ أحد الأوامر التى تستخدم إجراء مخزن. ويفترض هذا المثال، أننا قد قمنا بتهيئة مجموعة Parameters باستخدام ثلاثة معاملات هى: au_fname ،au_lname ،au_id.

```
With OleDbCommand1
  .CommandText = "UpdateAuthor"
  .CommandType = System.Data.CommandType.StoredProcedure
  .Parameters ("au_id").Value = listAuthorID.Text
  .Parameters ("au_lname").Value = txtAuthorLName.Text
  .Parameters ("au_fname").Value = txtAuthorFName.Text
End With
OleDbConnection1.Open ()
OleDbCommand1.ExecuteNonQuery ()
OleDbConnection1.Close ()
```

تقوم الإجراءات المخزنة فى الغالب بتمرير قيم فى الاتجاه المعاكس إلى التطبيق الذى قام باستدعائها. ويمكن أن يتم ذلك عن طريق تمرير القيمة باستخدام معامل أو عن طريق تعريف وتمرير القيمة العائدة (Return Value). للحصول على القيم التى يعيدها الإجراء

المخزن باستخدام معامل:

١. نقوم بإنشاء معاملات مع جعل خاصية Direction بها تساوى Output أو InputOutput. ونتحقق من أن نوع بيانات المعامل يتوافق مع القيمة العائدة.

٢. بعد تنفيذ الإجراء، نقرأ خاصية Value فى المعامل الذى يتم تمريرة.

الحصول على القيمة التى يعيدها الإجراء المخزن باستخدام القيمة العائدة (Return Value):

١. نقوم بإنشاء معامل ونضبط خاصية Direction على ReturnValue. ويجب ملاحظة أن يكون كائن Parameter الخاص بالقيمة العائدة هو البند الأول فى مجموعة Parameters.

٢. نتحقق من أن نوع بيانات المعامل يتوافق مع نوع القيمة العائدة المتوقعة. ومن الملاحظ أن عبارة Insert، عبارة Update، عبارة Delete تعيد لنا قيمة من نوع Integer تشير إلى عدد السجلات المتأثرة بعد تنفيذ العبارة. ويمكن الحصول على هذه القيمة فى صورة قيمة عائدة من وسيلة ExecuteNonQuery.

المثال التالى، يبين كيفية الحصول على القيمة العائدة من إجراء مخزن يدعى CountAuthors. ويفترض أن المعامل الأول فى مجموعة Parameters الخاصة بالأمر، يسمى "retvalue" وقد تم تهيئة اتجاهها باستخدام القيمة ReturnValue.

```
Dim cntAffectedRecords As Integer
OleDbcommand1.CommandText = "CountAuthors"
OleDbCommand1.CommandType = CommandType.StoredProcedure
OleDbConnection1.Open ()
OleDbCommand1.ExecuteNonQuery ()
OleDbConnection1.Close ()
cntAffectedRecords = CType (OleDbCommand1.Parameters ("retvalue").Value,
Integer)
MessageBox.Show ("Affected records = " &.cntAffectedRecords.ToString)
```

تنفيذ أوامر البيانات

بعد تهيئة خصائص أمر البيانات، يمكننا تنفيذ الأمر. وهناك أربعة وسائل يمكن استخدامها فى التنفيذ. ويعتمد استدعاء إحدى هذه الوسائل على العبارة أو الإجراء الذى سوف يتم تنفيذه وعلى النتائج المتوقعة من عملية التنفيذ. يمكن أن ينتج عن التنفيذ واحدا

من النتائج التالية:

- الحصول على مجموعة سجلات (Result Set). فى هذه الحالة، نقو بتنفيذ عبارة SQL أو إجراء مخزن ينتج عنة سجل أو أكثر. ويمكن الحصول على سجلات المجموعة واحدا فى كل مرة باستخدام قارئ بيانات (Data Reader).
- الحصول على عدد السجلات المتأثرة. يرتبط ذلك بتنفيذ الأوامر التى تقوم بتحديث قاعدة البيانات أو تغيير هيكل قاعدة البيانات.
- الحصول على قيمة فردية (Scalar Value). يخص هذا النوع من التنفيذ الإجراءات المخزنة أو عبارات SQL التى تقوم بالبحث أو حساب قيمة متوسطة على سبيل المثال.
- الحصول على بيانات فى صيغة XML. وهى إمكانية يدعمها SQL Server الإصدار ٧ أو ما بعدة.

تنفيذ أمر بيانات يعيد مجموعة من السجلات

يمكن استخدام أمر بيانات لتنفيذ أمر على قاعدة بيانات يعيد مجموعة من السجلات للقراءة فقط. يعنى ذلك تنفيذ عبارة SQL SELECT أو إجراء مخزن يحتوى على عبارة SELECT. يماثل ذلك ما يحدث عند تعبئة فئة بيانات باستخدام موفق بيانات، فيما عدا أن مجموعة السجلات يتم إعادتها مباشرة إلينا. ويمكن استخدام هذا النوع من أوامر البيانات فى الأغراض التالية:

- إعادة مجموعة من السجلات التى يمكن وضعها فى أداة تحكم على نموذج.
- إعادة سجل منفرد، يتم تحديده بدقة عن طريق تمرير معاملات إلى الأمر.

يقوم أمر البيانات بتنفيذ قراءة مجموعة من السجلات ووضعها فى قارئ بيانات من نوع OleDbDataReader أو نوع SqlDataReader. ثم يجرى استخدام حلقة لتكرار قراءة كل سجل على حدة بواسطة قارئ البيانات. ونظرا لأن قارئ البيانات يقوم بالقراءة فقط وللأمام، لذلك يتميز بالسرعة. لتنفيذ أمر يعيد مجموعة سجلات، نتبع الخطوات التالية:

١. نضيف أمر بيانات إلى النموذج.

٢. نضبط خاصية CommandText بكائن أمر البيانات على عبارة SQL أو اسم إجراء مخزن يعيد مجموعة سجلات.

٣. نضبط خاصية CommandType على CommandType.Text عند اختيار عبارة SQL أو على CommandType.Storedprocedure بالنسبة للإجراء المخزن.

٤. إذا كان الأمر يتطلب معاملات، يجب ضبطها.

٥. نقوم بتكوين Data Reader عن طريق تكوين مثل من تصنيف OleDbDataReader أو تصنيف SqlDataReader بناءا على نوع الاتصال وأمر البيانات المستخدم.

٦. نفتح الاتصال المرتبط بأمر البيانات.

٧. نستدعى وسيلة ExecuteReader بأمر البيانات، ونربط النتيجة بقارئ البيانات السابق تكوينه فى الخطوة الخامسة. ويجب ملاحظة أن وسيلة ExecuteReader تسمح لنا بتمرير معامل يحدد سلوك الأمر، شاملا خيارات إقفال الاتصال مباشرة، وإعادة سجل منفرد فقط، وإعادة مفاتيح فقط.

٨. نستخدم وسيلة Read فى قارئ البيانات للقراءة المتكررة. فى حلقة إلى أن يصبح العائد من هذه الوسيلة يساوى False.

٩. نقوم بإقفال قارئ البيانات.

١٠. نقوم بإقفال الاتصال إذا لم يتم ذلك من خلال استدعاء وسيلة ExecuteReader.

المثال التالى، يوضح كيفية قراءة السجلات من جدول Authors فى قاعدة بيانات Pubs الموجودة فى SQL Server. يخصص المثال القيمة TableDirect للخاصية CommandType، كما يضبط خاصية CommandText على اسم جدول Authors، ويمكن أيضا ضبط هاتين الخاصتين باستخدام نافذة Properties. وعند استدعاء وسيلة ExecuteReader، يقوم المثال بتكرار تنفيذ الوسيلة داخل حلقة بقارئ البيانات وبناء سلسلة محددة تتكون من معلومات عن المؤلف وعرض هذه السلسلة فى مربع نص. ويتم فى هذا المثال، استخراج بيانات العمود بالإشارة إلى اسم العمود فى قارئ البيانات.

```
Dim dreader As System.Data.OleDb.OleDbDataReader
OleDbCommand1.CommandText = "authors"
```

```
OleDbCommand1.CommandType = CommandType.TableDirect
OleDbConnection1.Open ()
dreader = OleDbCommand1.ExecuteReader (CommandBehavior.CloseConnection)
Dim s As String = ""
While dreader.Read ()
  s &= dreader("au_id").ToString() _
    & vbTab & dreader ("au_fname").ToString () _
    & " " & dreader ("au_lname").ToString & ControlChars.CrLf
End While
TextBox1.Text = s
dreader.Close ()
```

ويوضح المثال التالى، طريقتين لاستخدام قارئ البيانات للحصول على البيانات. الجزء الأول يماثل المثال السابق، ولكنة يستخدم قارئ بيانات لتحميل مربع سرد(ListBox) فى نموذج ويندوز بعمود author_id فى جدول authors. الجزء الثانى هو معالج حدث SelectedIndexChanged بمربع السرد. حيث تستخدم هذه الوسيلة قيمة au_id التى يختارها المستخدم لتكون معاملا لعبارة SQL SELECT لإعادة سجل واحد لأن قارئ البيانات متوقع منة أن يعيد سجل واحد. وبالنظر إلى أن قارئ البيانات سوف يعيد سجل واحد، لذلك لا نحتاج إلى استخدام حلقة معالجة، وبدلا من ذلك يتم استدعاء وسيلة Read مرة واحدة. ومن الملاحظ أن المثال لا يقوم بالتحقق من أن هناك على الأقل سجل واحد فقط عن طريق استدعاء وسيلة Read داخل عبارة If.

```
Public Class Form1
   Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

Private Sub Button1_Click (ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnLoadListBox.Click
        Dim dreader As System.Data.OleDb.OleDbDataReader
        OleDbCommand1.CommandText = "authors"
        OleDbCommand1.CommandType = CommandType.TableDirect
        OleDbConnection1.Open ()
        dreader = OleDbCommand1.ExecuteReader ()
        While dreader.Read ()
           ListBox1.Items.Add (dreader ("au_id"))
        End While
```

```
        dreader.Close ()
        OleDbConnection1.Close ()
End Sub
Private Sub ListBox1_SelectedIndexChanged (ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles ListBox1.SelectedIndexChanged
        Dim dreader As System.Data.OleDb.OleDbDataReader
        OleDbCommand2.CommandText = "SELECT au_id, au_lname, au_fname
from
        Authors WHERE au_id = ?"
        OleDbCommand2.CommandType = CommandType.Text
        OleDbCommand2.Parameters ("au_id").Value = ListBox1.Text
        OleDbConnection1.Open ()
        dreader = OleDbCommand2.ExecuteReader ()
        If dreader.Read () Then
           txtAuthorID.Text = dreader ("au_id").ToString ()
           txtAuthorLastName.Text = dreader ("au_lname").ToString ()
           txtAuthorFirstName.Text = dreader ("au_fname").ToString ()
        End If
        dreader.Close ()
        OleDbConnection1.Close ()
   End Sub
End Class
```

تحديث قواعد البيانات باستخدام أوامر البيانات

بعض أنواع أوامر البيانات التى نقوم بتنفيذها على قاعدة البيانات لا تعيد أى قيمة فيما عدا قيمة تشير إلى نجاح الأمر. تشمل هذه الأنواع من الأوامر:

- أوامر تعريف قاعدة البيانات (Database Definition Commands) التى نستخدمها لإنشاء وإدارة هياكل قاعدة البيانات مثل الجداول والإجراءات المخزنة.
- أوامر التحديث (Update Commands) التى تشمل عبارات INSERT، UPDATE، DELETE.

ويجب أن نستخدم أوامر قاعدة البيانات لتنفيذ عمليات تعريف هياكل البيانات(DDL) التى تعتبر الطريقة الوحيدة للقيام بهذه العمليات فى ADO.NET. وعند استخدام فئة بيانات، لا نحتاج إلى استخدام أمر بيانات منفصل لتنفيذ عمليات التحديث بقاعدة البيانات. وبدلا من ذلك، نستخدم موفق البيانات لتحديث قاعدة البيانات. وفى حالة عدم

استخدام فئات البيانات، يمكن إرسال أوامر التحديث مباشرة إلى قاعدة البيانات. وتعيد الأوامر المذكورة قيمة من نوع Integer تشير إلى نجاح العملية. وتختلف القيمة العائدة بناءا على ما إذا كنا نقوم بتحديث السجلات أو بتعريف هيكل بيانات (DDL):

- عند إنشاء أو تعديل هياكل قاعدة بيانات، تكون القيمة العائدة (-1) عندما تكون العملية ناجحة.
- عند تحديث السجلات، القيمة العائدة تشير إلى عدد السجلات التى تأثرت بالعملية.
- عندما تفشل العملية فى كل من الحالتين، تساوى القيمة العائدة صفر.

لتنفيذ أمر بيانات على قاعدة بيانات:

١. نضيف أمر بيانات إلى النموذج ونقوم باعدادة باستخدام عبارة SQL أو إجراء مخزن.
٢. عندما يتطلب الأمر معاملات، نقوم بتهيئتها.
٣. إذا كان هناك معاملات، نضيف كود لضبط قيم المعاملات.
٤. نضيف الكود اللازم لفتح الاتصال المرتبط مع أمر البيانات.
٥. نستدعى وسيلة ExecNonQuery. ومع أن هذه الوسيلة تعيد قيمة وحيدة من نوع Integer، يمكن أن يعيد الإجراء المخزن الذى قد نستعمله مجموعة معاملات خاصة بالأمر.
٦. نغلق الاتصال.

المثال التالى، يبين كيفية إنشاء جدول جديد فى قاعدة بيانات. ويفترض هذا المثال أننا قمنا بإضافة أمر بيانات واتصال إلى النموذج أو المكون وأن الأمر تم تهيئته لاستخدام الاتصال. ويقوم الكود بضبط خاصية CommandText على عبارة SQL Server Create Table التى تقوم بإنشاء جدول من عمودين وتعيد هذه الوسيلة القيمة (-1) عند النجاح فى التنفيذ وإنشاء الجدول.

```
Dim newtablecmd As String
```

```
Dim cmdresults As Integer
newtablecmd = "CREATE TABLE LookupCodes (code_id  smallint IDENTITY (1,1)
PRIMARY KEY CLUSTERED, code_desc  varchar (50) NOT NULL)"
OleDbCommand1.CommandType = CommandType.Text
OleDbCommand1.CommandText = newtablecmd
OleDbConnection1.Open ()
cmdresults = OleDbCommand1.ExecuteNonQuery ()
OleDbConnection1.Close ()
MessageBox.Show ("After creating the table, results = " & cmdresults.ToString)
```

ويبين المثال التالى، كيفية استخدام أمر بيانات لتحديث قاعدة بيانات عن طريق إدراج سجل جديد فى جدول Authors بقاعدة بيانات SQL Server Pubs. فى هذا المثال، يقوم الأمر باستدعاء إجراء مخزن يسمى "NewAuthor" مع افتراض أنة يحتوى على عبارة SQL Server INSERT. ويفترض المثال أن عبارة INSERT تحتوى على تسعة قيم لإنشاء السجل الجديد. وقد تم تهيئة أمر البيانات OleDbCommand2 لكى يحتوى على تسعة معاملات لمقابلة القيم المطلوبة لتكوين السجل الجديد. ويستخدم الكود مربعات نص فى نموذج لضبط قيم المعاملات، فتح الاتصال، استدعاء وسيلة ExecNonQuery، ثم إغلاق الاتصال بعد الانتهاء من التنفيذ.

```
Dim cmdresults As Integer
OleDbcommand2.CommandText = "NewAuthor"
OleDbCommand2.CommandType = CommandType.StoredProcedure
OleDbCommand2.Parameters ("au_id").Value = TextBox1.Text
OleDbCommand2.Parameters ("au_lname").Value = TextBox2.Text
OleDbCommand2.Parameters ("au_fname").Value = TextBox3.Text
OleDbCommand2.Parameters ("phone").Value = TextBox4.Text
OleDbCommand2.Parameters ("address").Value = TextBox5.Text
OleDbCommand2.Parameters ("city").Value = TextBox6.Text
OleDbCommand2.Parameters ("st").Value = TextBox7.Text
OleDbCommand2.Parameters ("zip").Value = TextBox8.Text
OleDbCommand2.Parameters ("contract").Value = CheckBox1.Checked
OleDbConnection2.Open ()
Try
  cmdresults = OleDbcommand2.ExecuteNonQuery ()
Catch ex as Exception
  MessageBox.Show ("Failed to execute command, err = " & ex.Message)
End Try
OleDbConnection2.Close ()
```

```
MessageBox.Show ("Number of records inserted = " & cmdresults.ToString)
```

تنفيذ أمر بيانات يعيد قيمة مفردة

قد نحتاج أحيانا إلى تنفيذ إجراء أو دالة على قاعدة بيانات تعيد قيمة مفردة. وبسبب إعادة قيمة واحدة، لا يتم تنفيذ هذا النوع من الأوامر مع فئة بيانات. المثال النموذجي على ذلك هو عبارة SQL التى تعيد قيمة تجميعية، مثل عبارة SUM، عبارة COUNT. ومن الأمثلة الأخرى، الإجراء المخزن الذى يأخذ كود المنتج ويعيد إلينا اسم المنتج. لتنفيذ أمر من هذا النوع، ننفذ الخطوات التالية:

١. نضيف أمر بيانات إلى النموذج.
٢. نضبط خاصية CommandText لتحديد عبارة SQL أو الإجراء المخزن الذى يجب تنفيذه.
٣. نضبط خاصية CommandType على CommandType.Text بالنسبة لعبارات SQL وعلى CommandType.StoredProcedure بالنسبة للإجراء المخزن.
٤. إذا كان الأمر يتطلب معاملات، نقوم بضبط هذه المعاملات.
٥. نقوم بفتح الاتصال المرتبط مع أمر البيانات.
٦. نستدعى وسيلة ExecuteScalar ونربط النتيجة مع متغير من نوع البيانات المناسب.
٧. نغلق الاتصال.

المثال التالى، يبين كيفية استخدام أمر بيانات لتنفيذ عبارة SQL تعيد قيمة عددية. تقوم عبارة SELECT فى هذا المثال باستعلام حول منتجات، وتتطلب إدخال معامل ProductID، وتعيد قيمة صحيحة تشير إلى كمية المخزون لهذا المنتج.

```
Dim scalarcmd As String
Dim qtyinstock As Integer
scalarcmd = "SELECT UnitsInStock FROM Products WHERE  ProductID = ? "
OleDbCommand3.CommandType = CommandType.Text
OleDbCommand3.CommandText = scalarcmd
OleDbCommand3.Parameters ("productID").Value = txtProductID.Text
OleDbConnection1.Open ()
qtyinstock = CType (OleDbCommand3.ExecuteScalar (), Integer)
```

```
OleDbConnection1.Close ()
MessageBox.Show ("QtyinStock = " & qtyinstock.ToString)
```

فئات البيانات

يمكن تعريف فئة البيانات (datasets) بأنها حاوية أو ذاكرة وسيطة لتخزين البيانات التى يستعملها التطبيق. وتعتبر فئة البيانات جزءا أساسى فى بناء تقنية ADO.NET. وتوفر فئات البيانات أداء مرتفع خاص بالوصول إلى البيانات، كما توفر المقدرة على التوسع (Scalability). ويماثل هيكل فئة البيانات هيكل قاعدة البيانات العلاقية (Relational Database)، لأنها تحتوى على هيكل من الجداول، الصفوف، والأعمدة. بالإضافة إلى أنها تحتوى على قيود وعلاقات يتم تعريفها. وتستخدم فئات البيانات عندما نريد العمل مع مجموعة من الجداول والصفوف أثناء عدم الاتصال مع مصدر البيانات. ومع أن فئات البيانات تمثل الوضع الأمثل للوصل إلى البيانات فى معظم الحالات، إلا أن هناك حالات لا يعتبر استخدام فئات البيانات هو الوضع الأمثل فيها. ويمكن تكوين ومعالجة فئات البيانات باستخدام التصنيفات التالية الموجودة فى نظام NET Framework:

- تصنيف فئة البيانات (DataSet Class)
- تصنيف جدول فئة البيانات (DataTable Class)
- تصنيف عمود بيانات (DataColumn Class)
- تصنيف قيود البيانات (Constraint Class)
- تصنيف علاقات البيانات (DataRelation Class)
- تصنيف صف البيانات (DataRow Class)

وتعتبر فئات البيانات التى نقوم باستخدامها أمثلة من تصنيف فئة البيانات (DataSet Class). ويتم الكشف عن الأجزاء الضرورية فى كائنات فئات البيانات من خلال هياكل البرمجة القياسية، مثل الخصائص (Properties)، والمجموعات (Collections).

تكوين فئات البيانات

يمكننا تكوين فئات البيانات باستخدام عدد من الوسائل، مثل الوسائل المرئية التى

يوفرها Visual Studio، أو استخدام الكود فى استنساخ أمثلة من تصنيف فئة البيانات وضبط الخصائص المتعلقة بها. ويستلزم تكوين فئات البيانات بطريقة صحيحة، الإحاطة بالمفاهيم المتعلقة بها.

المفاهيم المتعلقة بفئات البيانات

يرتبط بكائنات فئات البيانات (datasets) عدد من المفاهيم، مثل مخططات فئات البيانات، صيغة XML، فئات البيانات النوعية وغير النوعية، تأهيل فئات البيانات، موقع السجل والتجول فى فئة البيانات، العلاقات بين الجداول وكائنات العلاقات، القيود المفروضة فى فئات البيانات، تحديث فئات ومصادر البيانات.

تعريف مخططات فئات البيانات باستخدام صيغة *XML*

يستخدم نظام NET Framework كود XML لتخزين البيانات وتكوين مخططاتها. والمقصود بالمخطط هنا الجداول والأعمدة التى تتكون منها فئة البيانات. وتستخدم XML لبناء هذه المخططات على أساس المعايير القياسية التى وضعها World Wide Web Consortium لتعريف هياكل بيانات XML. ويمكن قراءة مخططات فئات البيانات باستخدام وسيلة ReadXMLSchema، كما يمكن كتابة هذه المخططات باستخدام وسيلة WriteXMLSchema. وعندما لا يكون المخطط متاحا، يمكن أن تقوم فئة البيانات باستنتاجه من البيانات الموجودة فى مستند XML به بيانات علاقية عن طريق استخدام وسيلة InferXMLSchema. وأهم المزايا المترتبة على استخدام XML فى فئات البيانات هى:

- يمكن قراءة مستند أو تيار XML فى فئة بيانات باستخدام وسيلة ReadXML وكتابة محتويات فئة بيانات فى صيغة XML باستخدام وسيلة WriteXML. وبالنظر إلى أن صيغة XML تعتبر قياسية لتبادل المعلومات بين التطبيقات المختلفة، لذا يمكننا تأهيل فئة بيانات بالمعلومات المرسلة فى صيغة XML بواسطة التطبيقات الأخرى. وبالمثل، يمكن كتابة محتويات فئة بيانات فى صورة مستند أو تيار XML لاستخدامه بواسطة التطبيقات الأخرى أو لتخزينها فى صورة قياسية.

يمكن تكوين مشهد XML من محتويات فئة بيانات، ثم مشاهدة ومعالجة البيانات باستخدام الوسائل العلاقية الموجودة فى فئة البيانات أو وسائل XML. وعند إحداث

تغييرات فى أحد المشهدين، مشهد فئة البيانات أو مشهد XML، فإنها تنعكس على المشهد الأخر

فئات البيانات النوعية وغير النوعية

يمكن أن تكون فئة البيانات نوعية أو غير نوعية. ترتبط فئة البيانات النوعية بمخطط داخلي يعبر عن هيكل بنائها وهى مثل من تصنيف يتم بناؤه باستخدام تصنيف DataSet الموجود بالنظام، بالإضافة إلى استخدام البيانات المتوفرة فى ملف XML امتدادة (xsd). وفى التصنيف الجديد، يتم التعبير عن جداول وأعمدة فئة البيانات فى صورة مجموعة من الكائنات والخصائص.

وعلى العكس من ذلك، لا ترتبط فئة البيانات غير النوعية بمخطط بيانات داخلي. وكما فى فئة البيانات النوعية، تحتوى فئة البيانات غير النوعية على جداول، أعمدة، وغيرها من المكونات. ويمكن استخدام أى من نوعى فئة البيانات فى تطبيقاتنا. إلا أن Visual Studio يحتوى على أدوات أكثر لتدعيم فئات البيانات النوعية، مما يجعل برمجة فئات البيانات النوعية أكثر سهولة واقل أخطاء.

الوصول إلى البيانات فى فئات البيانات النوعية وغير النوعية

يتم التعامل مع الجداول والأعمدة فى فئات البيانات النوعية على أنها كائنات. وعلى هذا الأساس يمكن الحصول على قيمة عمود CustomerID الموجود فى جدول Customers بفئة البيانات النوعية باستخدام الكود التالى:

```
Dim s As String
s = dsCustomersOrders1.Customers (0).CustomerID
```

على العكس من ذلك، عند العمل مع فئة بيانات غير نوعية يكون الكود المقابل كما يلى:

```
Dim s As String
s = CType(dsCustomersOrders1.Tables
("Customers").Rows(0).Item("CustomerID"), String)
```

تأهيل فئات البيانات

كما سبق إيضاحه، تعتبر فئة البيانات حاوية يجب تعبئتها بالبيانات قبل استخدامها. ويمكن تأهيل أو تعبئة فئة البيانات باستخدام عدد من الطرق:

- استدعاء وسيلة Fill فى موفق البيانات. يؤدى ذلك إلى قيام الموفق بتنفيذ عبارة SQL أو الإجراء المخزن (Stored Procedure) لاستخراج النتائج من قاعدة البيانات ووضعها فى فئة البيانات. وعندما تحتوى فئة البيانات على العديد من الجداول، يمكن أن يكون هناك موفق بيانات منفصل لكل جدول، وبناءا على ذلك، يجب استدعاء وسيلة Fill فى كل موفق بيانات على حدة.
- تأهيل جداول فئة البيانات يدويا عن طريق إنشاء كائنات DataRow وإضافتها إلى مجموعة Rows بالجدول. ويمكن القيام بذلك فقط أثناء تشغيل التطبيق.
- قراءة مستند أو تيار XML فى فئة البيانات.
- نسخ محتويات فئة بيانات أخرى. ويمكن أن يكون هذا السيناريو مفيدا عندما يحصل التطبيق على فئات البيانات من مصادر مختلفة، ولكنة يحتاج إلى دمجها فى فئة بيانات واحدة.

تحديد موقع السجل فى فئة البيانات

بالنظر إلى أن فئة البيانات تعتبر حاوية بيانات منفصلة بالكامل عن مصدر البيانات، لذا نجد أن فئات البيانات (datasets) على العكس من فئات السجلات (recordsets)، لا تدعم مفهوم السجل الجاري (Current Record). وبدلا من ذلك، تكون جميع السجلات فى فئة البيانات متاحة للاستخدام. وبالنظر إلى عدم وجود سجل جارى، لذا لا توجد خاصية تشير إلى السجل الجاري استخدامه، كما لا توجد وسائل أو خصائص للحركة من سجل إلى أخر. على هذا الأساس، يمكننا الوصول إلى الجداول المنفردة فى فئة البيانات بصفتها كائنات. ويكشف لنا كل جدول فى فئة البيانات عن مجموعة من الصفوف، التى يمكن الوصول إلى الصفوف التى تحتويها باستخدام العبارات الخاصة بالمجموعة.

الجداول المرتبطة وكائنات العلاقات

عندما يكون لدينا العديد من الجداول فى فئة بيانات، يمكن أن تكون المعلومات الموجودة فى هذه الجداول متعلقة ببعضها. ولا يوجد فى فئة البيانات معرفة موروثة خاصة بهذه العلاقات، لذلك يجب إنشاء كائنات تمثل هذه العلاقات (DataRelation). تقوم هذه

الكائنات بوصف العلاقات بين الجداول فى فئة البيانات. ويمكن استخدامها فى استخراج السجلات التابعة لسجل أصلى، أو العكس. على سبيل المثال، بالمقارنة مع قاعدة بيانات Northwind فى SQL Server، يمكن أن تحتوى فئة البيانات على جدول عملاء به السجلات التالية:

CustomerID	CompanyName	City
ALFKI	Alfreds Futterkiste	Berlin
ANTON	Antonio Moreno Taquerias	Mexico D.F.
AROUT	Around the Horn	London

كما يمكن أن تحتوى فئة البيانات أيضا، على جدول أخر به معلومات أمر مبيعات. ويحتوى سجل أمر المبيعات على عمود CustomerID، الذى يمثل مفتاح خارجي (Foreign Key). ويمكن أن يظهر جدول أوامر المبيعات على الصورة التالية:

OrderId	CustomerID	OrderDate
10692	ALFKI	10/03/1997
10702	ALFKI	10/13/1997
10365	ANTON	11/27/1996
10507	ANTON	4/15/1997

يوجد بين هذه الجدولين علاقة واحد إلى كثير (One-to-many)، نظرا لأن كل عميل يمكن أن يتبعة أكثر من أمر مبيعات. بناءا على هذه العلاقة، يمكن استخدام كائنات العلاقات (DataRelations) للحصول على السجلات ذات العلاقة من الجدول التابع أو الجدول الأصلى.

قيود الوصول إلى البيانات

كما هو الحال فى معظم قواعد البيانات، تدعم فئات البيانات استخدام القيود (Constraints) للتحقق من تكامل البيانات والمحافظة على هذا التكامل. ويمكن تعريف القيود بأنها قواعد يجرى تطبيقها عند إدراج الصفوف، تحديث الصفوف، أو حذف الصفوف فى جدول بفئة البيانات. ويمكن تمييز نوعين من القيود:

- قيود التفرد (Unique Constraint). تقوم هذه القيود بالتحقق من تفرد القيم الجديدة فى عمود بأحد الجداول.

- قيود المفتاح الخارجي (Foreign Key Constraint). تحدد هذه القيود العلاقة التى تحكم كيفية تحديث السجلات التابعة عند تحديث أو إلغاء سجل فى الجدول الأصلى.

وترتبط القيود فى فئة البيانات بالجداول أو ترتبط بأعمدة الجداول. ويتم تكوين القيود فى صورة كائنات من نوع UniqueConstaint أو نوع ForeignKeyConstraint. ثم يتم بعد ذلك إضافتها إلى مجموعة القيود (Constraints Collection) بالجدول. وبدلا من ذلك، يمكن تحديد قيد التفرد بضبط خاصية Unique فى أحد الأعمدة على قيمة True. وتدعم فئة البيانات خاصية EnforceConstraints التى تتحكم فى فرض القيود أو عدم فرضها. قيمة هذه الخاصية تساوى True فى الوضع الإفتراضى. ومع ذلك، هناك أوقات يكون من المفيد فيها تغيير قيمة هذه الخاصية إلى False. يحدث ذلك فى الغالب، عندما نريد تغيير أحد السجلات بطريقة مخالفة للقيود الموضوعة ثم إعادة القيود إلى الوضع الطبيعى بعد ذلك.

فى Visual Studio، يتم إنشاء القيود ضمنيا عند تعريف فئة البيانات. عند إضافة مفتاح أساسى إلى فئة البيانات، يتم ضمنيا إنشاء قيد تفرد خاص بعمود المفتاح الأساسى. ويمكن تكوين قيود تفرد للأعمدة الأخرى عن طريق ضبط خاصية Unique الخاصة بالعمود على القيمة True. ويتم تكوين مفاتيح خارجية عن طريق تكوين كائنات DataRelation فى فئة البيانات.

تحديث فئات ومصادر البيانات

عند تغيير السجلات فى فئة بيانات، يجب كتابة هذه التغييرات فى الاتجاه المعاكس، بمعنى كتابتها فى قاعدة البيانات. ولكتابة التغييرات التى تحدث فى فئة البيانات بقاعدة البيانات، نستدعى وسيلة Update بموفق البيانات الذى يصل ما بين فئة البيانات وبين مصدر البيانات المقابل. وتستخدم خاصية RowState فى تصنيف DataRow لمعرفة الصفوف التى يجب استخدامها لتحديث قاعدة البيانات. تشير هذه الخاصية إلى حالة تغيير صف البيانات وكيفية هذا التغيير منذ تحميل جدول البيانات لأول مرة من قاعدة البيانات. وقد تشتمل هذه الخاصية على القيم التالية: Modified، Deleted، Unchanged. ولتحديث قاعدة البيانات، تقوم وسيلة Update بفحص قيمة خاصية

RowState لتحديد السجلات التى يجب كتابتها فى قاعدة البيانات وتحديد أمر قاعدة البيانات المناسب لتنفيذ العملية.

وسائل تكوين فئات البيانات

هناك عدد من الوسائل التى يوفرها لنا Visual Studio لإنشاء فئات البيانات:

- مصمم المكونات (The Component Designer)، الذى يظهر اسفل مصمم النماذج. يستقبل هذا المصمم عناصر البيانات التى يتم سحبها إلى النموذج من مربع ToolBox أو مربع Server Explorer. ويمكن عن طريق هذه الوسيلة تكوين فئات البيانات النوعية وغير النوعية.
- مصمم XML، الذى يسمح بتكوين مخطط فئة البيانات وبنائها على أساس هذا المخطط.
- استخدام الكود لتكوين فئات بيانات ديناميكية غير نوعية.

تكوين فئات البيانات باستخدام Component Designer

عندما يكون النموذج فى وضع التصميم، يظهر مربع Component Designer تلقائيا أسفل مصمم النماذج عند إضافة مكون أو أداة غير مرئية إلى نموذج. ويمكن التمييز بين طريقتين لتكوين فئات البيانات باستخدام هذه الأداة. الطريقة الأولى تقوم على أساس استخدم موفق البيانات (Data Adapters) فى تكوين فئة البيانات، والطريقة الثانية تستخدم مكون فئة البيانات (DataSet Component) للقيام بهذه المهمة.

يترتب على استخدام موفق البيانات فى تكوين فئة البيانات، تكوين فئة بيانات نوعية على أساس المخطط الذى يحدده موفق البيانات. الخطوات التالية توضح كيفية استخدام موفق البيانات فى تكوين فئة بيانات:

1. نضيف واحدا أو أكثر من موفقات البيانات إلى النموذج ونقوم بإعدادها.
2. فى مصمم المكونات أسفل مصمم النماذج، نختار موفق البيانات الذى سوف يستخدم فى تحويل البيانات بين مصدر البيانات وبين فئة البيانات. من الناحية النموذجية نستخدم موفق بيانات لكل جدول. ولذلك يجب اختيار كل موفقات

البيانات الخاصة بكل الجداول التى نريد العمل معها لكى يمكن تكوين فئة بيانات تحتوى على العديد من جداول البيانات.

٣. من قائمة Data، نختار Generate Dataset. يترتب على ذلك، عرض مربع حوار Generate Dataset.

٤. نختار New ثم نحدد اسم فئة البيانات. وعندما نرغب فى إضافة فئة البيانات إلى النموذج، ننقر Add an instance of this DataSet to the designer. يستخدم Visual Studio المعلومات الموجودة فى موفق البيانات لتكوين ملف مخطط XML (.xsd)، ويتم إضافة هذا الملف إلى المشروع باستخدام الاسم السابق تحديده. ويقوم Visual Studio بعد ذلك، بتكوين ملف تصنيف فئة بيانات جديد بناءا على المخطط الذى تم تكوينه.

٥. نضيف الكود اللازم لتأهيل فئة البيانات التى يتم تكوينها. ويتم ذلك عن طريق استدعاء وسيلة Fill بموفق البيانات.

٦. عندما نريد تحديث فئة البيانات أو قاعدة البيانات، نضيف كود إلى النموذج للقيام بذلك.

ويتيح لنا مكون DataSet الموجود فى مربع ToolBox، تكوين فئات البيانات النوعية التى ترتبط بمخطط داخلي للبيانات (Built-in Schema)، كما يتيح لنا تكوين فئات البيانات غير النوعية التى لا تحتوى على مخطط بيانات خاص بها. لتكوين فئة بيانات بهذه الطريقة، نتبع الخطوات التالية:

١. من صفحة ملصق Data بمربع ToolBox، نسحب مكون DataSet إلى النموذج. يترتب على ذلك ظهور مربع حوار Choose a Dataset المبين بالشكل رقم (١٥).

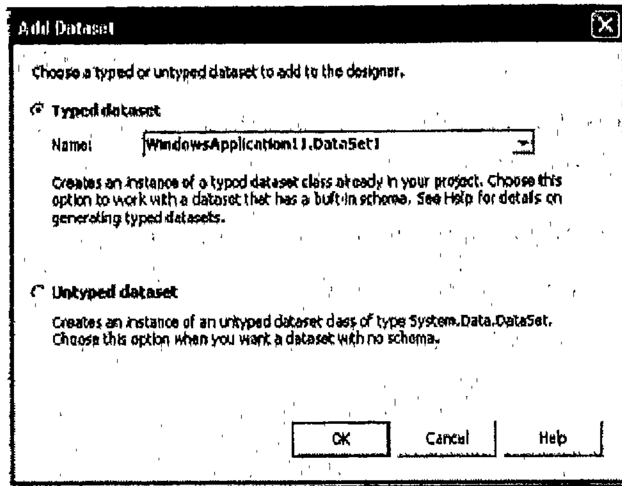
٢. وعندما نريد تكوين فئة بيانات نوعية مشتقة من تصنيف فئة بيانات نوعية موجود، نختار Typed Dataset.

٣. عندما نريد تكوين فئة بيانات غير نوعية، نختار Untyped Dataset ثم ننقر Ok. يؤدى ذلك إلى إضافة فئة بيانات خالية إلى النموذج.

٤. إذا كنا نرغب فى تخصيص اسم جديد لفئة البيانات، نختار الفئة ثم نفتح نافذة Properties ونقوم بتدقيق كل من الخاصتين التاليتين:

أ. خاصية Name التى تحدد اسم الكائن الذى نستخدمه فى الكود للإشارة إلى فئة البيانات.

ب. خاصية DataSetName التى تقوم بتحديد اسم فئة البيانات الذى نستخدمه عند تصديرها إلى مستند XML.



شكل رقم ١٥

تكوين مخططات فئات البيانات باستخدام مصمم XML

نستخدم مصمم XML عندما نريد التحكم بطريقة أكثر دقة فى كيفية تعريف مخطط فئة البيانات أو عندما لا نستطيع تكوين المخطط من مصدر البيانات. ويمكن استخدام مصمم XML لتكوين مخطط فئة بيانات بطريقتين:

- سحب عناصر قاعدة البيانات من مربع Server Explorer إلى المصمم. يترتب على ذلك، تكوين المخطط الناتج من هيكل قاعدة البيانات، والذي يمكن تعديله عند الحاجة.
- تكوين ملف مخطط بأنفسنا. ويعتبر ذلك مفيدا عند تصميم مخطط بدون الإشارة إلى

مصدر بيانات خارجي. في كل من الحالتين، يجرى تكوين تصنيف DataSet بناءا على المخطط الذى تم تكوينه واستخدامه على النموذج. ويسمح لنا مصمم XML بإضافة، حذف، أو تعديل عناصر. كما يسمح لنا أيضا بالقيام بوظائف، مثل إضافة كائن DataRelation التى لايمكن القيام بها باستخدام Component Designer. وعند استخدام مصمم XML، فإننا فى الواقع نقوم بتحرير ملف xsd. وعند حفظ التغييرات فى هذا الملف، يتم إعادة تكوين ملف التصنيف المقابل ليعكس المخطط بعد تحديثه.

صيانة فئات البيانات

بعد تكوين فئات البيانات، قد نحتاج إلى تعديل هياكلها بإضافة عناصر متنوعة إليها. كما نحتاج إلى إضافة فئات البيانات بأنواعها إلى النماذج. تتناول البنود التالية موضوعات متنوعة تشمل إضافة جداول وأعمدة وعلاقات وقيود إلى فئات البيانات غير النوعية، إضافة أعمدة تحتوى على تعبيرات حسابية إلى فئات البيانات النوعية وغير النوعية، إضافة جداول جديدة إلى فئة بيانات قائمة، وإضافة فئات البيانات إلى النماذج.

إضافة جداول وأعمدة إلى فئة البيانات غير النوعية

لإضافة جداول وأعمدة إلى فئة البيانات غير النوعية، نستخدم محرر المجموعة (Collection Editor) فى نافذة Properties. لتنفيذ ذلك، نتبع الخطوات التالية:

١. نختار فئة البيانات على النموذج ونفتح نافذة Properties.

٢. ننقر الزر الموجود فى خاصية Tables لفتح مربع حوار Tables Collection Editor.

٣. ننقر Add لإضافة جدول جديد. وفى حالة الرغبة فى إعادة تسمية الجدول، نقوم باختياره فى القسم الأيمن من شبكة Properties ثم نغير قيمة خاصية Name.

٤. لإضافة أعمدة إلى الجدول، نتبع الخطوات التالية:

أ. نتحقق من اختيار الجدول فى قائمة Members، ثم ننقر الزر الموجود فى خاصية Column لعرض مربع حوار Columns Collection Editor.

ب. ننقر Add لتكوين عمود جديد، ثم نضبط الخصائص الخاصة به فى الشبكة

التى على يمين مربع الحوار. وتسمح لنا بعض الخصائص بضبط القيود المتعلقة ببيانات الأعمدة، مثل قيد التفرد أو قيد عدم قبول القيمة null.

ت. نكرر الخطوة (ب) لكل عمود نريد إضافته.

ث. عند الانتهاء من إضافة الأعمدة الخاصة بالجدول، ننقر Ok لإقفال مربع الحوار.

٥. لتحديد مفتاح أساسى للجدول، نستخدم القائمة المنسدلة فى خاصية PrimaryKey على النحو التالى:

أ. فى القائمة المنسدلة، نختار مربع الاختيار التالى لكل جزء من أجزاء المفتاح الأساسى.

ب. بعد اختيار الأعمدة التى تكون المفتاح الأساسى، يمكننا توسيع خاصية PrimaryKey وضبط خصائص كل عمود فى المفتاح الأساسى. ويتم ضبط خاصية Unique تلقائيا لكل الأعمدة التى تكون المفتاح الأساسى.

٦. نكرر الخطوات من ٣ إلى ٥ لكل جدول ونضبط الأعمدة التى نريد إضافتها إلى فئة البيانات.

إضافة القيود إلى فئة بيانات غير نوعية

يمكن تعريف نوعين من القيود فى فئة البيانات:

- قيد التفرد (Unique Constraint)، الذى يقوم بالتحقق من أن القيم الجديدة فى الأعمدة غير متكررة فى الجدول.
- قيد المفتاح الخارجي (Foreign-key Constraint)، الذى يعرف القواعد التى تحكم تحديث سجلات الجدول التابع عند تحديث أو حذف سجل فى الجدول الأصلى.

لإضافة قيود إلى فئة بيانات غير نوعية فى وقت التصميم:

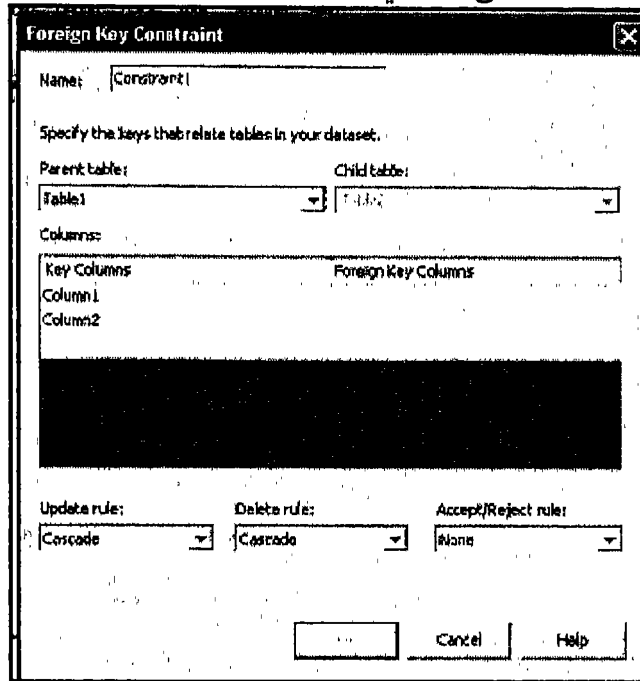
١. نختار فئة البيانات على النموذج، ثم نفتح نافذة Properties.

٢. ننقر الزر الموجود فى خاصية Tables لفتح مربع حوار Tables Collection Editor.

٣. نختار الجدول الذى نريد إضافة قيود إلية، ثم ننقر الزر الموجود فى خاصية

Constraints لفتح مربع حوار Constraints Collection Editor.

٤. ننقر Add ثم نختار Unique Constraint أو Foreign Key Constraint من القائمة المنسدلة. يترتب على ذلك عرض مربع الحوار ذات العلاقة. وللمساعدة فى تعبئة مربعات الحوار المذكورة، يمكننا الضغط على مفتاح F1. يبين الشكل رقم (١٦) مربع حوار إضافة مفتاح خارجي بهذه الطريقة.



شكل رقم ١٦

إضافة العلاقات إلى فئة بيانات غير نوعية

يمكننا إضافة كائنات علاقات (DataRelation Objects) بين الجداول فى فئة بيانات أثناء وقت التصميم، باتباع الخطوات التالية:

١. نختار فئة البيانات على النموذج ونفتح نافذة Properties.

٢. ننقر الزر الموجود فى خاصية Relations لفتح مربع حوار DataRelation Collection Editor.

٣. ننقر Add لإنشاء علاقة جديدة. يترتب على ذلك ظهور مربع حوار DataRelation Editor.

٤. نضبط خصائص العلاقة المطلوبة.

٥. ننقر Ok لحفظ تغييرات العلاقة.

٦. نكرر الخطوات من ٣ إلى ٥ لكل علاقة نريد تكوينها.

تكوين أعمدة فئة بيانات باستخدام التعبيرات

يمكن أن يحتوى جدول البيانات فى فئة البيانات على أعمدة تكون القيمة بها نتيجة لعملية حسابية بدلا من قراءتها من مصدر بيانات. على سبيل المثال، فى سجل أمر المبيعات يكون من المناسب أكثر، تعريف عمود يحتوى على القيمة الكلية لكمية عن طريق ضرب الكمية فى سعر الوحدة بدلا من تخزين هذا الإجمالي فى سجل. ويمكن أن تقوم الأعمدة المحسوبة أيضا بإضافة أو عد القيم فى سجلات تابعة وترشيح السجلات الأخرى. ولتعريف الأعمدة المحسوبة، نستخدم العبارات (Expressions) التى تقوم بتحديد طريقة حساب هذه الأعمدة.

ويتكون التركيب اللغوى للتعبير من علامات حسابية، منطقية، وسلاسل وقيم. ويمكن الإشارة إلى قيمة البيانات باستخدام اسم العمود الخاص بها واستخدام الدوال التجميعية، مثل Count. على سبيل المثال، التعبير الخاص بعمود القيمة الإجمالية المبنى على عمود سعر الوحدة وعمود الكمية، يمكن يماثل ما يلى:

UnitPrice * Quantity

ونشير إلى الأعمدة فى جدول تابع عن طريق استخدام الكلمة الإرشادية "Child" متبوعة باسم العمود. على سبيل المثال، ينتج عن التعبير التالى متوسط القيم الموجودة فى عمود السعر فى كل السجلات الموجودة فى جدول تابع:

Avg (Child.Price)

لتكوين العمود والتعبير فى فئة بيانات نوعية باستخدام مصمم XML، نتبع الخطوات التالية:

١. إذا لم يكن المخطط مفتوحا فى مصمم XML، نقوم بالنقر المزدوج على ملف المخطط

(.xsd) فى مربع Solution Explorer.

٢. فى الشبكة التى تمثل الجدول المطلوب العمل معه، نضيف عمود جديد عند طريق تنفيذ الخطوات التالية:

أ. فى السطر الأول الخالي من الشبكة، نختار element من القائمة المنسدلة.

ب. فى المربع التالى على نفس السطر، ندخل اسم العمود باستخدام تعريف صحيح.

ت. فى المربع التالى، نختار نوع بيانات مناسب للعملية الحسابية.

٣. نختار العمود الجديد فى شبكة الجدول، ثم ندخل تعبير فى خاصية Expression فى نافذة Properties.

ولتكوين عمود والتعبير الخاص به فى فئة بيانات غير نوعية، نتبع الخطوات التالية:

١. نختار فئة البيانات غير النوعية على النموذج ثم نفتح نافذة Properties.

٢. ننقر الزر الموجود فى خاصية Tables لفتح مربع حوار DataTable Collection Editor.

٣. فى محرر المجموعة (Collection Editor)، ننقر الزر الموجود فى خاصية Columns لفتح مربع حوار DataColumn Collection Editor.

٤. ننقر Add لإضافة عمود جديد.

٥. نختار العمود الجديد ثم نضبط خاصية ColumnName فى نافذة Properties لتحديد اسم العمود.

٦. نختار نوع بيانات مناسب لنتيجة العملية الحسابية.

٧. نقوم بإدخال التعبير المناسب فى خاصية Expression.

إضافة الجداول إلى فئات بيانات قائمة

قد نحتاج أحيانا إلى إضافة جدول إلى فئة بيانات سبق تكوينها. إحدى الطرق التى يمكن استخدامها لتنفيذ ذلك، هى حذف فئة البيانات ثم إعادة تكوينها لتشتمل على

الجداول الجديدة. غير أن هذه الطريقة قد تتسبب فى فقد بعض المعلومات الخاصة بمكونات فئة البيانات، مثل القيود (Constraints)، وكائنات علاقات البيانات. وتشمل الطرق الأخرى ما يلى:

- استخدام Component Designer. ويعتبر ذلك حلا سهلا يسمح لنا بسحب عناصر إلى النموذج ثم إعادة تكوين فئة البيانات المعدلة.
- استخدام مصمم XML لتعديل مخطط فئة البيانات. يعتبر هذا الأسلوب مفيدا عندما يكون هناك تعود على استخدام مصمم XML وفى نفس الوقت لا نحتاج أو لا نرغب فى فتح نموذج.

لإضافة جدول إلى فئة بيانات قائمة، نستخدم الخطوات التالية:

١. نضيف موفق بيانات (Data Adapter) يمثل الجدول الذى نريد إضافته إلى النموذج.
٢. نختار موفق البيانات، ثم نختار Generate Dataset من قائمة Data فى القائمة الرئيسية. يترتب على ذلك عرض مربع حوار Generate Dataset.
٣. تحت Choose a dataset، نختار Existing ثم اسم فئة البيانات التى نريد إضافة الجدول إليها.
٤. تحت Choose which table(s) to add to the dataset، نختار أسماء جميع الجداول التى نريد إضافتها إلى فئة البيانات.
٥. إذا كنا نرغب فى إضافة مثل من فئة البيانات المعدلة على النموذج، نختار Add an instance of this to the designer.
٦. ننقر Ok لتكوين فئة البيانات المحدثة. يترتب على ذلك إضافة معلومات عن الجدول المضاف إلى مخطط فئة البيانات والى ملف تصنيف فئة البيانات.

إضافة فئة بيانات نوعية إلى نموذج

سوف نحتاج دائما إلى إضافة فئة بيانات نوعية إلى نموذج لكى يمكن التعامل مع البيانات، ربط أدوات التحكم مع فئة البيانات، وغير ذلك من المهام. وعندما نقوم بإضافة

فئة بيانات، فإن ذلك يعنى إضافة مثل من تصنيف فئة البيانات النوعية إلى النموذج. ويجب أن يكون تصنيف فئة البيانات موجودا بالفعل. ويمكن أن يأتى هذا التصنيف من عدة مصادر:

- سبق تكوينه فى نموذج أو مكون أخر بالمشروع.
- سبق تكوينه يدويا فى مصمم XML.
- عن طريق إنشاء مرجع إلى مكون أخر ينتج عنة فئة بيانات.

لإضافة فئة بيانات نوعية إلى نموذج، نطبق الخطوات التالية:

١. نفتح النموذج الذى نريد العمل معه.
٢. من صفحة ملصق Data بمربع ToolBox، نسحب كائن DataSet إلى المصمم. يترتب على ذلك، ظهور مربع حوار Choose a Dataset.
٣. نختار Typed Dataset، ثم نختار فئة البيانات التى نريد استخدامها من القائمة المنسدلة ثم ننقر Ok.
٤. نضيف الكود اللازم إلى النموذج لتأهيل فئة البيانات. وبصفة عامة، يتم استدعاء وسيلة Fill الموجودة فى موفق البيانات لتنفيذ عبارة SQL أو إجراء مخزن لاستخراج البيانات من فئة البيانات، كما يتضح من الكود التالى:

```
OleDbDataAdapter1.Fill( dsCustomersOrders1, "Customers" )
```

٥. إذا كان التطبيق يستلزم كتابة معلومات من فئة البيانات إلى قاعدة البيانات، نضيف الكود اللازم إلى النموذج للقيام بذلك. نستدعى وسيلة Update لكتابة التغييرات فى قاعدة البيانات، عند تعبئة فئة البيانات باستخدام موفق بيانات.

إضافة فئة بيانات غير نوعية إلى نموذج

فئة البيانات غير النوعية هى مثل من تصنيف DataSet الموجود فى نظام NET Framework. وعلى خلاف فئة البيانات النوعية، لا تمثل فئة البيانات غير النوعية مثل من تصنيف ناتج عن استخدام ملف مخطط فئة بيانات (xsd.). ولذلك لا يوجد بفئة

البيانات غير النوعية هيكل موروث. فى فئات البيانات غير النوعية، نقوم بتكوين الجداول، الأعمدة، القيود، وكائنات العلاقات إثناء وقت التصميم باستخدام نافذة Properties، أو أثناء وقت التشغيل باستخدام الكود، أو أن ندع ذلك لموفق البيانات الذى يقوم بعملية تعبئة فئة البيانات. ومع أن فئة البيانات غير النوعية لا تستخدم تصنيفا مرتبطا بمخطط، إلا أنة يمكننا بعد تكوين فئة البيانات غير النوعية، استيراد هيكلها من ملف مخطط فئة بيانات.

فرز وترشيح البيانات

بعد تأهيل فئة بيانات، يكون من المفيد فى أغلب الأحوال العمل مع مجموعات فرعية مختلفة من سجلات الجدول، أو مشاهدة البيانات فى ترتيب مختلف. يمكن القيام بذلك من خلال إحدى الطرق التالية:

- استخدام مشاهد البيانات (DataView Objects). ويمكن تعريف مشهد البيانات بأنة كائن يمثل طبقة على قمة جدول البيانات، تقدم مشهد به سجلات جدول البيانات بعد ترشيحها وفرزها. ولا يعتبر مشهد البيانات نسخة من هذه البيانات، ولكنة طريقة مختلفة لرؤية البيانات الموجودة فى أحد الجداول.
- استخدام وسيلة Select فى الجدول لترشيح وفرز البيانات. ولا يترتب عليها هذه الطريقة تغيير محتويات أو ترتيب السجلات فى الجدول، ولكنها تعيد إلينا مصفوفة من السجلات المختارة على أساس المعيار المحدد فى وسيلة Select.

كل من الطريقتين توفر نفس إمكانيات الترشيح والفرز. والفرق الأساسى بينهما هو أننا لا نستطيع استدعاء وسيلة SELECT إلا عن طريق الكود فى وقت التشغيل. من ناحية أخرى، يوفر مشهد البيانات المزايا التالية:

- يمكن إنشاء وتهيئة مشهد البيانات أثناء وقت التصميم، مع توفر خيار ضبط الخصائص أثناء التشغيل.
- يمكن استخدام مشهد البيانات فى عمليات ربط البيانات مع أدوات التحكم.
- يمكن إنشاء مشاهد بيانات متعددة لرؤية البيانات بطرق مختلفة. على سبيل

المثال، يمكن أن يقوم أحد المشاهد بعرض بيانات جدول أوامر المبيعات مربعة على أساس التاريخ، بينما يقوم مشهد أخر بعرض هذه الأوامر مرتبة على أساس العميل.

استخدام مشاهد البيانات فى الفرز والترشيح

توجد كائنات مشاهد البيانات (DataView Objects) فى صفحة ملصق Data بمربع ToolBox. ولكى نستخدم هذه المشاهد، نضيف أحدها إلى النموذج بنفس الطريقة تقريبا التى نضيف بها عناصر البيانات الأخرى. يتم ذلك عن طريق سحب هذه المكونات إلى مصمم النماذج ثم ضبط خصائصها. وحتى فى حالة عدم إضافة مشهد بيانات مباشرة إلى النموذج، نجد أن هناك مشهد بيانات إفتراضى متاح لكل جدول فى فئة البيانات. يمكن الوصول إلى هذا المشهد الإفتراضى باستخدام خاصية DefaultView بالجدول. وفى معظم الأحوال، يمكننا استخدام مشهد البيانات الإفتراضى، غير أن استخدام مشاهد البيانات التى نقوم بإعدادها، يتميز بما يلى:

- يمكن استخدام مشاهد بيانات متعددة مع نفس البيانات.
- يمكن ضبط خصائص مشهد البيانات فى وقت التصميم، بينما يكون مشهد البيانات الإفتراضى متاحا فقط أثناء وقت التشغيل.
- يمكن تخصيص أسماء لمشاهد البيانات.

معيار الترشيح فى مشاهد البيانات

نستخدم خاصية RowFilter فى مشهد البيانات لتحديد المعيار الذى يتم على أساسه اختيار السجلات التى يشملها المشهد. وتستخدم نفس القواعد المستخدمة فى تكوين تعبيرات الأعمدة عند تكوين معايير الترشيح للبيانات. ويتم تقييم تعبير الترشيح على أساس أنة تعبير منطقى. فإذا كانت القيمة الناتجة من تقييم التعبير تساوى True، يتم إدراج السجل ضمن سجلات المشهد. ويمكن أن يأخذ التعبير الصورة التالية:

```
Price > 10.00
```

الترشيح على أساس حالة الصف ورقم الإصدار

يمكن الترشيح (Filtering) على أساس رقم إصدار السجل أو حالة السجل. يقوم الترشيح

برقم الإصدار على أساس أن فئة البيانات تحتفظ بإصدارات (Versions) متنوعة من السجلات فى الجدول الواحد. عندما يتم تعبئة فئة البيانات بالسجلات لأول مرة، سوف تحتوى الجداول على الإصدار الأصلى للسجلات. وعند تغيير أحد السجلات، تحتفظ فئة البيانات بإصدار مختلف من السجل يمثل الإصدار الحالى بجانب الإصدار الأصلى. ومن الاستخدامات الشائعة للمرشحات (Filters) تحديد الإصدارات الحالية من السجلات فى جدول بيانات. إذا كانت السجلات قد تم تغييرها، سوف يكون هناك إصدارين من السجلات: الإصدار الجاري الذى يعكس التغييرات التى حدثت والإصدار الأصلى الذى يمثل السجل قبل التعديل. ويعتمد الترشيح باستخدام حالة السجل على أساس أن السجلات فى جداول البيانات تحتفظ بإشارات تبين حالتها. هناك سجلات جديدة، سجلات لم تتغير، سجلات محذوفة، وسجلات معدلة.

فرز السجلات

يتشابه الفرز (Sorting) مع الترشيح فى أننا نحدد تعبير تتم على أساسه عملية الفرز. والتعبير النموذجي للفرز هو اسم العمود الذى يتم الفرز على أساسه. على سبيل المثال، عند الفرز على أساس عمود OrderDate، نحدد تعبير الفرز OrderDate. من ناحية أخرى، يمكن الفرز على أساس قيمة التعبير، بما فى ذلك قيمة العمليات الحسابية. ويتم فرز مشاهد البيانات عن طريق ضبط قيمة خاصية Sort على تعبير الفرز.

إضافة مشاهد بيانات إلى نموذج

نتبع الخطوات التالية لإضافة مشهد بيانات إلى النموذج:

١. نقوم بإنشاء فئة بيانات.
٢. نضيف الكود اللازم لتأهيل فئة البيانات.
٣. من صفحة ملصق Data بمربع ToolBox، نسحب كائن DataView إلى النموذج. يترتب على ذلك، إضافة مشهد بيانات باسم إفتراضى DataView1 إلى النموذج.
٤. عندما نرغب فى إعداد مشهد البيانات وقت التصميم، نختار المشهد ونستخدم نافذة Properties للقيام بهذا الإعداد.

استخدام مشاهد البيانات لترشيح وفرز البيانات

يسمح الفرز والترشيح باستخدام مشهد بيانات، بتحديد معيار الفرز أو الترشيح وقت التصميم ويوفر لنا كائن يمكن استخدامه للربط مع البيانات. ويمكن الترشيح والفرز باستخدام مشهد بيانات سبق إضافته إلى نموذج. القيام بذلك، يسمح لنا بضبط خيارات الترشيح والفرز فى وقت التصميم. على العكس من ذلك، يمكن استخدام مشهد البيانات الإفتراضى (Default View) المتاح تلقائيا لكل جدول فى فئة البيانات. وعند استخدام مشهد البيانات الإفتراضى، لا نستطيع ضبط خيارات الترشيح والفرز إلا باستخدام الكود. لترشيح وفرز مشهد بيانات :

١. عندما نرغب فى ضبط خيارات مشهد البيانات فى وقت التصميم، نضيف مشهد بيانات إلى النموذج.

٢. نضبط خاصية Sort فى مشهد البيانات باستخدام أحد تعبيرات الفرز (Sort Expression). ويمكن أن يشتمل تعبير الفرز على أسماء أعمدة جداول البيانات أو أعمدة عمليات حسابية. وعند الرغبة فى تحديد تعبير الفرز وقت التشغيل، فإن مشهد البيانات يعكس التعبير مباشرة.

٣. نضبط خاصية RowFilter بمشهد البيانات باستخدام تعبير ترشيح. ويجب أن ينتهى تقييم تعبير الترشيح إلى القيمة True أو القيمة False، كما يتضح من الكود التالى:

```
CustomerStatus = 'Active'
```

وللترشيح على أساس رقم الإصدار أو حالة السجل، نضبط خاصية RowStateFilter على قيمة من بين قيم تعداد DataViewRowState، كما يتضح من الكود التالى:

```
Dataview1.RowStateFilter = DataViewRowState.CurrentRows
```

يبين الكود التالى كيفية ضبط وفرز مشهد البيانات أثناء وقت التشغيل باستخدام مشهد البيانات الإفتراضى:

```
Dataset1.Customers.DefaultView.Sort = "City"
```

يفترض المثال التالى إضافة مشهد بيانات إلى النموذج، ويقوم باستخدام مشهد البيانات لفرز السجلات على أساس زر الراديو الذى يتم اختياره. كما يقوم باستخدام مشهد البيانات لترشيح السجلات الجارية فقط ثم عرض النتائج عن طريق الربط بين شبكة بيانات (Data Grid) وبين مشهد البيانات:

```
Public Class Form1
   Inherits System.Windows.Forms.Form
#Region " Windows Form Designer generated code "
Private Sub Form1_Load (ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
      SqlDataAdapter1.Fill (DataSet11, "Customers")
End Sub
Private Sub Button1_Click (ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
      DataView1.Table = DataSet11.Tables ("Customers")
      If radioCity.Checked Then
         DataView1.Sort = "City"
      ElseIf radioCompanyName.Checked Then
         DataView1.Sort = "CompanyName"
      End If
      DataView1.RowFilter = "Country = 'UK'"
      DataView1.RowStateFilter = DataViewRowState.CurrentRows
      Me.DataGrid1.DataSource = DataView1
End Sub
End Class
```

تكوين مدير مشاهد بيانات والعمل معه

مدير مشاهد البيانات هو عبارة عن كائن يحتوى على مجموعة من مشاهد البيانات، على أساس واحد لكل جدول فى فئة البيانات. ويوفر مدير مشاهد البيانات وصول مركزي إلى البيانات التى تم ترشيحها وفرزها. ويأخذ ذلك أهمية خاصة عند العمل مع الجداول ذات العلاقات. ويجب إنشاء وتهيئة مدير مشاهد البيانات باستخدام الكود بسبب عدم وجود كائن أثناء التصميم يمثل مدير مشاهد البيانات.

لتكوين وتهيئة مدير مشاهد بيانات، نتبع الخطوات التالية:

١. نقوم بتكوين فئة بيانات.

٢. نضيف الكود اللازم لتأهيل فئة البيانات.

٣. نضبط خاصية DataSet بمدير مشاهد البيانات على فئة البيانات السابق تكوينها.

٤. نحدد تعبيرات الفرز والترشيح بالوصول إلى الجداول المنفردة من خلال مجموعة DataViewSettings. ويجب الإنتباة إلى أن محاول الوصول إلى جدول باستخدام هذه الخاصية، يتسبب عنة قيام مدير مشاهد البيانات تلقائيا بتكوين مشهد بيانات للجدول.

٥. وعندما نريد ربط أدوات تحكم بمدير مشاهد بيانات، نقوم بضبط خصائص ربط البيانات بعد الانتهاء من تهيئة مدير مشاهد البيانات. على سبيل المثال، في نموذج ويندوز، يمكن ضبط خاصية DataSource على مدير مشاهد البيانات، وضبط خاصية DataMember على اسم مشهد البيانات المطلوب الربط معه.

المثال التالي، يبين كيفية إنشاء مدير مشاهد بيانات وتحديد معلومات الفرز والترشيح. يفترض المثال أن هناك فئة بيانات باسم dataset11 تحتوى على ثلاث جداول: Customers، Orders، و Products. كما يفترض وجود نموذج به شبكة بيانات(DataGrid) باسم DataGrid1 وثلاث أزرار راديو باسم RadioButton1، RadioButton2، RadioButton3 . بعد الانتهاء من تكوين مدير مشاهد البيانات، يقوم المثال بفرز مشهد بيانات Customers ومشهد بيانات Orders ثم يقوم بربط شبكة البيانات مع أحد أعضاء مدير مشاهد البيانات بناءا على اختيار أحد أزرار الراديو.

```
Dim dvm As New DataViewManager ()
SqlDataAdapter1.Fill (DataSet11, "Customers")
SqlDataAdapter2.Fill (DataSet11, "Orders")
SqlDataAdapter3.Fill (DataSet11, "Products")
dvm.DataSet = DataSet11
dvm.DataViewSettings ("Customers").RowFilter = "Country='Germany'"
dvm.DataViewSettings ("Customers").Sort = "CompanyName"
dvm.DataViewSettings ("Orders").Sort = "OrderDate"
DataGrid1.DataSource = dvm
If RadioButton1.Checked = True Then
   DataGrid1.DataMember = "Customers"
ElseIf RadioButton2.Checked = True Then
   DataGrid1.DataMember = "Orders"
```

```
Else
   DataGrid1.DataMember = "Products"
End If
```

الفرز والترشيح المباشر لجداول البيانات

يمكن ترشيح وفرز البيانات في الجداول مباشرة عن طريق استدعاء وسيلة Select بالجدول. تسمح هذه الطريقة بالترشيح والفرز في وقت التشغيل فقط. وعندما نقوم بالفرز مباشرة في جدول البيانات، لا نقوم بإعادة ترتيب محتويات الجدول، بل نعمل مع مجموعة تمثل نتائج فرز السجلات.

للترشيح والفرز مباشرة في جدول بيانات، نستدعى وسيلة Select بجدول البيانات، مع تمرير ثلاثة معاملات إلى هذه الوسيلة:

```
datatable.Select (filterExp, sortExp, rowstatevalue)
```

يمثل المعامل الأول في الكود السابق، التعبير المستخدم في ترشيح البيانات. ويمثل المعامل الثاني تعبير فرز البيانات. ويمكن أن يكون هذا التعبير اسم عمود أو تعبير حسابي ينتج عنة قيمة. ويمثل المعامل الأخير إصدار أو حالة السجل التي يمكن الفرز على أساسها. ويمكن استدعاء هذه الوسيلة باستخدام المعامل الأول فقط أو المعامل الأول والثاني. ولتجاوز أحد المعاملات، نمرر سلسلة خالية في موقع المعامل. وتعيد هذه الوسيلة مصفوفة من كائنات الصفوف (DataRow Objects).

المثال التالي،يبين كيفية ترشيح وفرز جدول Customers في فئة البيانات DataSet1. في هذا المثال، يتم الترشيح على أساس دولة العميل. كما يتم الفرز على أساس عمود City. ويتم عرض قائمة السجلات الناتجة في مربع قائمة سرد مفترض وجودها.

```
Dim filterExp As String = "Country = 'Germany'"
Dim sortExp As String = "City"
Dim drarray () As DataRow
Dim i As Integer
drarray = DataSet11.Customers.Select (filterExp, sortExp, _
DataViewRowState.CurrentRows)
For i = 0 To (drarray.Length - 1)
   ListBox1.Items.Add (drarray (i)("City").ToString)
Next
```

ويبين المثال التالى كيفية استدعاء وسيلة Select للترشيح على أساس حالة الصف،مع التجاوز عن المعامل الأول والمعامل الثانى. وينتج عن التنفيذ إعادة مصفوفة تشتمل على السجلات المحذوفة.

```
Drarray = dataSet1.Customers.Select (Nothing, Nothing, _
DataViewRowState.Deleted)
```

استخدام سجلات مشاهد البيانات

عند العمل مع مشهد بيانات، يمكن الوصول إلى السجلات التى تم ترشيحها أو فرزها عن طريق الحصول عليها من مشهد البيانات بدلا من الجدول مباشرة. ويمكن أيضا تحديث ، إدراج، وحذف سجلات من خلال استخدام مشهد البيانات، مع وجود بعض التحفظات. ويجب أن يشتمل مشهد البيانات على معلومات كافية عن كل سجل لكى يمكن تحديد أين يذهب هذا السجل فى جدول البيانات. يمكن أن تشمل هذه المعلومات المفتاح الأساسى، أو أعمد أخرى توفر معلومات كافية عن السجل. ولكى يمكن تنفيذ أنواع العمليات المختلفة على مشهد البيانات، يجب ضبط خصائص AllowDelete، AllowNew، AllowEdit بمشهد البيانات على القيمة True لكل نوع من العمليات المقابلة.

العثور على السجلات فى مشاهد البيانات

للعثور على السجلات فى مشاهد البيانات، ننفذ الخطوات التالية:

١. نضبط خاصية Sort بمشهد البيانات لكى تشمل العمود أو الأعمدة التى نريد البحث على أساسها.

٢. نستدعى وسيلة Find أو FindRows بمشهد البيانات، مع تمرير القيمة التى نريد البحث على أساسها فى أعمدة الفرز.

أ. عندما نرغب فى العثور على سجل واحد، نستدعى وسيلة Find.

ب. عندما نريد العثور على أكثر من سجل، نستخدم وسيلة FindRows.

المثال التالى يبين كيفية العثور على الفهرس الخاص بالسجل بناءا على القيمة الموجودة فى مربع نص:

```
DataView1.Sort = "CustomerID"
Dim foundIndex as Integer = DataView1.Find (TextBox1.Text)
```

قراءة السجلات فى مشهد بيانات

للوصول إلى سجل فى مشهد بيانات، نستخدم فهرس السجل، ويمكن الوصول إلى عمود باستخدام الاسم. المثال التالى، يقوم بالحصول على اسم العميل (Customer name) من السجل الأول فى مشهد بيانات:

```
Dim DataView1 as New DataView (ds.Customers)
Dim cname As String = CType (DataView1 (0)("CompanyName"), String)
```

ويبين المثال التالى حلقة لمعالجة سجلات مشهد بيانات يستخدم جدول عملاء. فى كل سجل من السجلات، يقوم المثال بالحصول على القيم من ثلاثة حقول: CompanyName، CustomerID، City. يقوم المثال بعد ذلك بوصل جميع البيانات فى سلسلة محددة، وعرض السلسلة فى مربع نص:

```
Dim drv As DataRowView
Dim s As String = ""
For Each drv In DataView1
  s &= drv("CustomerID").ToString & "   "
  s &= drv("CompanyName").ToString & "   "
  s &= drv("City").ToString & ControlChars.CrLf
Next
TextBox1.Text = s
```

تحديث سجلات مشهد بيانات

نستخدم الفهرس لتحديد السجل المطلوب تحديثه، ثم نضبط قيمة العمود عن طريق استخدام الاسم للإشارة إلية. ويجب ملاحظة أننا لا نستطيع تدقيق السجلات من خلال مشهد بيانات، إذا لم تكن قيمة خاصية AllowEdit تساوى True. يوضح المثال التالى كيفية التعرف على وتحديث أحد الأعمدة فى مشهد بيانات:

```
DataView1 (0)("CompanyName") = "Fabrikam, Inc."
```

إدراج السجلات فى مشهد بيانات

١. نستدعى وسيلة AddNew بمشهد البيانات. تقوم هذه الوسيلة بتكوين سجل جديد وينتج عنها كائن DataRowView.

٢. نقوم بتحديث السجل كما نفعل عند تحديث أى سجل من خلال مشهد بيانات.

ويجب ملاحظة أننا لا نستطيع إدراج سجلات من مشهد بيانات إذا لم تكن قيمة خاصية AllowNew تساوى True.

المثال التالى، يوضح كيفية إضافة سجل جديد إلى مشهد بيانات، تحديث ثلاثة حقول به:

```
Dim drv As DataRowView
drv = DataView1.AddNew ()
drv ("CustomerID") = "AAA"
drv ("CompanyName") = "AAFabrikam, Inc."
drv ("City") = "Aurora"
```

حذف السجلات من خلال مشهد البيانات

لحذف سجل من خلال مشهد بيانات، نستدعى وسيلة Delete بمشهد البيانات، ونمرر إليها فهرس السجل المطلوب حذفة. ولا يمكن حذف سجل من مشهد بيانات، إذا لم تكن قيمة خاصية AlloDelete تساوى True. الكود التالى يوضح طريقة حذف السجل:

```
DataView1.Delete (0)
```

العلاقات بين البيانات

فى كثير من الأوقات تحتاج تطبيقاتنا إلى العمل مع الجداول ذات العلاقات. ومع أن فئة البيانات تحتوى على جداول وأعمدة كما فى قاعدة البيانات، إلا أنها لا تشتمل على مقدرة قاعدة البيانات الخاصة بتحديد العلاقات بين الجداول. ولذلك يتيح لنا النظام إمكانية تكوين كائنات العلاقات (DataRelation Objects) لتمكيننا من تكوين العلاقات بين الجداول الأصلية والجداول التابعة بناءا على مفتاح رئيسي. على سبيل المثال، فئة البيانات التى تحتوى على معلومات المبيعات، يمكن أن تحتوى على جدول للعملاء (Customers) وجدول لأوامر المبيعات (Orders). وحتى إذا كانت الجداول تحتوى على مفتاح مشترك، فإن الجداول ذاتها لا تحتفظ بسجل عن صفوف أحد الجداول التى تحتفظ بعلاقة مع صفوف جدول أخر.

كائنات العلاقات

تستخدم كائنات العلاقات (DataRelation Objects) للاحتفاظ بالعلاقات بين الجداول الأصلية والجداول التابعة. وتمثل جزءا من هيكل فئة البيانات بجانب الجداول

والأعمدة المعرفة فى فئة البيانات. ويمكن تحديد الوظائف التى يقوم بها كائن العلاقات فيما يلى:

- تتيح كائنات البيانات استخدام السجلات المرتبطة بسجل أخر يجرى استخدامه. وتوفر السجلات التابعة عند العمل مع سجل أصلى، كما توفر السجل الأصلى عند العمل مع سجل تابع.
- يمكن استخدامها لفرض قيود ضمان التكامل المرجعي (Referential Integrity)، مثل حذف السجلات التابعة عندما نقوم بحذف سجل أصلى.

ومن المهم فهم الفرق بين الوصل الحقيقي (True Join) وبين الوظيفة التى يقوم بها كائن DataRelation. فى الوصل الحقيقي، يتم وضع سجلات الجداول الأصلية وسجلات الجداول التابعة فى مجموعة سجلات واحدة بملف غير مفهرس. ولا يترتب على استخدام كائنات البيانات تكوين مجموعة سجلات جديدة، ولكنها تقوم بتتبع العلاقة بين الجداول والمحافظة على السجلات الأصلية والسجلات التابعة متزامنة.

الوصول إلى السجلات ذات العلاقات

عندما تكون هناك علاقات منطقية بين الجداول فى فئة بيانات، يمكن أن يقوم كائن DataRelation بجعل السجلات ذات العلاقة الموجودة فى جدول أخر متاحة للاستخدام. على سبيل المثال، إذا كان لدينا جدول عملاء يحتوى على مفتاح رئيسي هو CustomerID وجدول أوامر مبيعات به مفتاح خارجي هو CustomerID، يمكن استخدام كائن علاقة بين الجدولين وضبط خصائصه ليعكس هذه المفاتيح. يمكن بعد ذلك استخدام كائن العلاقة للحصول على السجلات التى بينها علاقات، عن طريق تمريرة من خلال وسيلة GetChildRows الموجودة فى كائن DataRow بالجدول الأصلى. وينتج عن تنفيذ هذه الوسيلة الحصول على مصفوفة تحتوى على السجلات التابعة. المثال التالى يوضح كيفية الحصول على السجلات التابعة. فى هذا المثال، يتم ضبط مصفوفة drarray لكى تحتوى على السجلات التابعة للصف الأول فى جدول العملاء (Customers Table):

```
Dim RowCtr As Integer
Dim drarray () As DataRow
RowCtr = 0
```

```
drarray =
dsCustomersOrders1.Customers(RowCtr).GetChildRows("CustomersOrders")
```

ويمكن أيضا الحصول على السجل الأصلى باستخدام سجل تابع عن طريق استدعاء وسيلة GetParentRow الموجودة فى كائن DataRow بالجدول التابع. وفى هذه الحالة لا ينتج عن تنفيذ هذه الوسيلة مصفوفة سجلات، ولكن ينتج صف واحد يمثل السجل الأصلى.

فرض القيود باستخدام كائنات العلاقات

يستخدم كائن العلاقة أيضا فى تكوين وفرض القيود التالية :

- قيد التفرد (Unique Constraint). يضمن هذا القيد عدم احتواء أعمدة جداول البيانات فى فئة البيانات على قيم متكررة.
- قيد مفتاح خارجي (Foreign-key Constraint). يستخدم للحفاظ على تكامل المراجع بين الجداول الأصلية والجداول التابعة فى فئة البيانات.

ويتم تنفيذ القيود التى نحددها فى كائن DataRelation عن طريق تكوين الكائنات المناسبة وضبط الخواص تلقائيا. عند تكوين قيد مفتاح خارجي (Foreign Key) باستخدام كائن DataRelation، يتم إضافة أمثلة من تصنيف ForeignKeyConstraint إلى خاصية ChildKeyConstraint بكائن DataRelation. ويتم تنفيذ قيد التفرد إما عن طريق ضبط خاصية Unique فى كائن DataColumn على القيمة True، أو عن طريق إضافة مثل من تصنيف UniqueConstraint إلى خاصية ParentKeyConstraint فى كائن DataRelation.

تكوين كائنات العلاقات

يمكن تكوين كائنات العلاقات وإضافتها إلى مجموعة DataRelationCollection فى فئة البيانات باستخدام الكود أو باستخدام مصمم XML.ويمكن الوصول إلى مجموعة DataRelationCollection من خلال خاصية Relations فى فئة البيانات، خاصية ChildRelations، خاصية ParentRelations فى كائن DataTable بفئة البيانات.

تكوين كائنات العلاقات باستخدام الكود

يقوم المثال التالى بتكوين علاقة جديدة ثم إضافتها إلى مجموعة العلاقات فى فئة البيانات:

```
Private Sub CreateRelation ()
```

- الإعلان عن كائن العمود الأصلى وكائن العمود التابع.

```
  Dim parentCol As DataColumn
  Dim childCol As DataColumn
```

- تخصيص أعمدة للمتغيرات السابق الإعلان عنها.

```
  parentCol = DataSet1.Tables ("Customers").Columns ("CustID")
  childCol = DataSet1.Tables ("Orders").Columns ("CustID")
```

- تكوين كائن العلاقة.

```
  Dim relCustOrder As DataRelation
  relCustOrder = New DataRelation ("CustomersOrders", parentCol, childCol)
```

- إضافة العلاقة إلى مجموعة العلاقات فى فئة البيانات.

```
  DataSet1.Relations.Add (relCustOrder)
End Sub
```

تكوين كائنات العلاقات باستخدام XML

ليس من الضروري معرفة كود XML أو معرفة صيغة مخطط فئة البيانات، لكى نستطيع تكوين كائنات العلاقات باستخدام مصمم XML. للقيام بذلك، نطبق الخطوات التالية:

١. نقوم بتكوين مخطط لفئة البيانات، إذا لم يكن موجودا.

٢. فى Solution Explorer، ننقر نقرا مزدوجا على المخطط (ملف xsd) لفتحة فى مصمم XML. يترتب على ذلك فتح المخطط فى مشهد Schema، وعرض كل جدول فى شبكة خاصة به.

٣. من صفحة ملصق XML Schema فى مربع ToolBox، نسحب كائن Relation إلى الجدول التابع. يؤدى ذلك إلى فتح مربع حوار Edit Relation المبين بالشكل رقم (١٧).

٤. عندما نرغب فى تغيير اسم كائن العلاقة، نكتب الاسم الجديد فوق القيمة الموجودة فى مربع Name.

٥. نتحقق من ضبط Parent Element على اسم الجدول الأصلى، وضبط Child Element على اسم الجدول التابع.

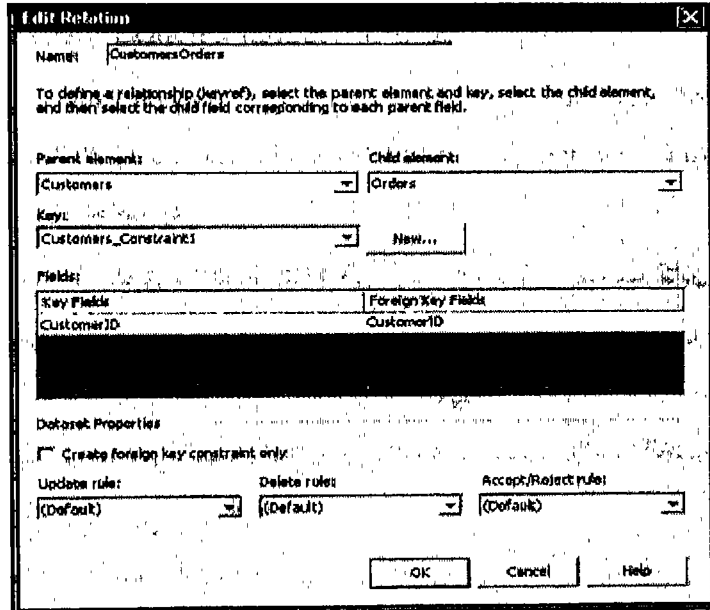
٦. من قائمة Key، نختار المفتاح الموجود فى الجدول الأصلى الذى نريد تكوين كائن DataRelation له. ويجب الإنتباة إلى أن الأسماء المعروضة فى قائمة Key ليست أسماء أعمدة المفاتيح، ولكنها عناصر يمكن أن تشير إلى أى عمود مرتبط بقيد تفرد (Unique Constraint).

٧. فى شبكة Fields، نتحقق من أن أعمدة المفتاح الأساسى فى الجدول الأصلى وعمود المفتاح الخارجي فى الجدول التابع قد تم ضبطها بطريقة مناسبة.

٨. يمكن اختيار Create Foreign Key Constraint Only عندما نرغب فى تكوين علاقة تستخدم فقط لفرض علاقة المفتاح الخارجي.

٩. يمكن اختيار القواعد التى تتحكم فى التحديث (Update rule) ، الحذف (Delete rule)، وعلاقة القبول والرفض (Accept/Reject rule) فى المربعات الخاصة بذلك.

١٠. ننقر Ok لتكوين كائن العلاقة. ويترتب على ذلك، قيام مصمم XML برسم خط العلاقة بين الجدول الأصلى وبين الجدول التابع مع تمثيل كائن العلاقة بمربع باهت اللون.



شكل رقم ١٧

ولتعديل كائن علاقة باستخدام مصمم XML، ننقر بزر الماوس الأيمن على كائن العلاقة فى مشهد Design بمصمم XML، ثم نختار Edit Relation. يترتب على ذلك فتح مربع حوار Edit Relation وعرض معلومات كائن العلاقة المطلوب تعديلها.

ولحذف كائن علاقة فى مصمم XML، ننقر كائن العلاقة فى مشهد Design، ثم نختار Delete من قائمة Edit.

تحديث فئات ومصادر البيانات

تتكون عملية تحديث مصدر البيانات من خطوتين. الخطوة الأولى هى تحديث فئة البيانات بالمعلومات الجديدة. والخطوة الثانية تتمثل فى إرسال التغييرات من فئة البيانات إلى مصدر البيانات الأصلى. ولا تقوم فئة البيانات بكتابة تغييرات البيانات تلقائيا فى مصدر البيانات الذى تعتمد علية، بل لابد من قيامنا صراحة بهذه الخطوة. ويتم تحقيق هذه الخطوة عن طريق استدعاء وسيلة Update الموجودة فى موفق البيانات المستخدم لتأهيل فئة البيانات. وتشتمل عمليات تحديث البيانات على إدراج السجلات، تغيير السجلات، وحذف هذه السجلات.

تحديث فئات البيانات

بعد تأهيل فئة البيانات، يتم فى الوضع المعتاد تنفيذ بعض أنواع المعالجة للبيانات قبل إرسالها إلى مصدر البيانات أو إلى برنامج أو تطبيق أخر. وحيث أن كل سجل فى فئة البيانات هو عبارة عن كائن DataRow، لذا يتم إنجاز التغييرات فى فئة البيانات عن طريق تحديث وحذف هذا الكائن. بالإضافة إلى ذلك، يمكن إدراج سجلات جديدة فى فئة البيانات عن طريق إضافة كائنات DataRow إلى مجموعة Rows فى كائن DataTable بفئة البيانات.

تحديث سجلات فئة بيانات

لكى يتم تدقيق سجل قائم فى فئة بيانات، نحتاج إلى الوصول إلى عمود بيانات معين فى صف محدد. ويمكن تحقيق ذلك فى كل من فئة البيانات النوعية وغير النوعية باستخدام إحدى طريقتين:

- عن طريق الفهارس في مجموعات الجداول، مجموعات الصفوف، ومجموعات الأعمدة.
- عن طريق تمرير اسم الجدول واسم العمود إلى المجموعات ذات العلاقة.

لتحديث السجلات الموجودة في فئات البيانات النوعية وغير النوعية، نخصص قيمة لعمود معين في كائن DataRow. ويلاحظ أن أسماء الجداول والأعمدة لا تكون متاحة في وقت التصميم بفئات البيانات النوعية، لذا يجب الوصول إليها باستخدام الفهارس الخاصة بها. المثال التالي، يوضح كيفية تحديث البيانات في أول عمودين بالسجل الخامس في الجدول الأول بفئة البيانات dataset1. ويتم الوصول إلى قيم البيانات باستخدام فهارس مجموعات الأعمدة، مجموعات الصفوف، ومجموعات الجداول:

```
dataset1.Tables (0).Rows (4).Item (0) = "Wingtip Toys"
dataset1.Tables (0).Rows (4).Item (1) = "Buffalo"
```

ويوضح المثال التالي كيفية تحديث نفس البيانات الموجودة في المثال السابق، مع استبدال فهارس المجموعات بأسماء الجداول والأعمدة التي يجرى تمريرها في صيغة سلاسل:

```
dataset1.Tables ("Customers").Rows (4).Item ("CompanyName") = "Wingtip Toys"
dataset1.Tables ("Customers").Rows (4).Item ("City") = "Buffalo"
```

إدراج سجلات جديدة في فئة بيانات

لكي يمكن إضافة سجلات جديدة إلى فئة بيانات، يجب تكوين صف بيانات جديد وإضافته إلى مجموعة DataRow في جدول البيانات. يوضح المثال التالي كيفية إدراج صفوف إضافية في كائن DataTable في فئة بيانات. ويفترض هذا المثال وجود جدول باسم ExistingTable في فئة بيانات وأن هذا الجدول به عمودين هما: عمود FirstName وعمود LastName.

لإضافة سجلات جديدة إلى فئة بيانات نوعية أو غير نوعية، ننفذ الخطوات التالية:

١. نستدعي وسيلة NewRow في كائن DataTable لإنشاء سجل جديد خالي. هذا السجل الجديد، يرث هيكل الأعمدة الخاص به من مجموعة DataColumnCollection:

```
Dim anyRow as DataRow = ExistingTable.NewRow
```

٢. نقوم بتحديث السجل الجديد باستخدام الكود التالي:

```
anyRow (0) = "Jay"
anyRow (1) = "Stevens"
```

أو الكود التالى:

```
anyRow ("FirstName") = "Jay"
anyRow ("LastName") = "Stevens"
```

٣. نضيف السجل الجديد إلى الجدول عن طريق استدعاء وسيلة Add فى كائن DataRowCollection:

```
ExistingTable.Rows.Add (anyRow)
```

إدراج سجلات جديدة فى فئة البيانات النوعية

تعرض فئات البيانات النوعية أسماء الأعمدة على أنها خصائص لكائن DataRow. يترتب على ذلك توفر إمكانية استخدام أسماء الأعمدة مباشرة، كما يتضح من الكود التالى:

```
Dim anyRow as DataRow = DatasetName.ExistingTable.NewRow
anyRow.FirstName = "Jay"
anyRow.LastName = "Stevens"
ExistingTable.Rows.Add (anyRow)
```

حذف السجلات من فئة بيانات

لكى يتم الاحتفاظ بالمعلومات التى تحتاجها فئة البيانات لتحديث مصدر البيانات بطريقة صحيحة، نستخدم وسيلة Delete لحذف صفوف جدول بفئة البيانات. وعندما لا يحتاج التطبيق إلى إرسال التغييرات إلى مصدر بيانات، يمكن حذف السجلات بالوصول المباشر إلى مجموعة صفوف البيانات (data row collection).

وعند حذف السجلات باستخدام وسيلة Delete، لا تقوم هذه الوسيلة بحذف السجل فيزيائيا ولكنها تضع إشارة حذف على السجل. بناءا على ذلك، تشتمل عملية عد السجلات باستخدام وسيلة Count على السجلات المحذوفة التى سبق وضع إشارة الحذف عليها. المثال التالى يوضح كيفية استدعاء وسيلة Delete لوضع إشارة الحذف على السجل الأول فى جدول Customers:

```
DsCustomers1.Customers.Rows (0).Delete ()
```

الأحداث المتعلقة بتحديث البيانات

عند تحديث سجلات فئة بيانات، يقع عدد من الأحداث التى ترتبط بكائن DataTable. هذه الأحداث، يمكن الاستجابة لها أثناء تنفيذ التغييرات وبعد الانتهاء منها.

يوضح الجدول رقم (٢٤) هذه الأحداث.

الحدث	الإيضاح
ColumnChanging	يحدث أثناء تغيير قيمة عمود
ColumnChanged	يحدث بعد تغيير قيمة العمود
RowChanging	يحدث أثناء تغيير القيم فى أحد الصفوف
RowChanged	يحدث بعد تغيير القيم فى أحد الصفوف
RowDeleting	تحدث فى مرحلة حذف الصف
RowDeleted	يحدث بعد الانتهاء من حذف الصف

جدول ٢٤

ونظرا لأن أحداث RowChanging، ColumnChanging، RowDeleting تقع أثناء عملية التحديث، لذا يمكن استخدامها فى مراجعة البيانات والتحقق من صحتها أو إنجاز أنواع أخرى من المعالجة. وحيث أن عملية التحديث تكون مستمرة أثناء وقوع هذه الأحداث، لهذا يمكن إلغاء التحديث بإصدار استثناء (Exception) يمنع إتمام عملية التغيير.

تعطيل قيود التحديث

إذا كانت فئة البيانات تحتوى على قيود، مثل المفاتيح الخارجية (Foreign Keys)، يمكن أن يترتب على تحديث عمود فى سجل حدوث مخالفة لهذة القيود. ويمكن أن يكون أحد الأعمدة فى حالة خطأ بعد تحديثه وقبل الانتقال إلى العمود التالى. على سبيل المثال، نفترض وجود جدول فى فئة بيانات يحتوى على عمودين، ولا يسمح أى منهما بإدخال قيمة من نوع null. بمجرد أن نبدأ فى إدخال سجل جديد، سوف يكون هناك دائما قيمة null فى عمود واحد على الأقل. فإذا لم تكن هناك آلية لتعطيل هذا القيد مؤقتا، سوف يصدر خطأ فى كل مرة ننتهي فيها من كتابة العمود الأول وقبل الانتقال إلى العمود الثانى.

لتعليق قيود التحديث، ننفذ الخطوات التالية:

١. قبل تغيير البيانات فى الصف، نستدعى وسيلة BeginEdit فى كائن DataRow. ويجب الإنتباة إلى أن التغييرات التى نحدثها فى الصف بعد استدعاء هذه الوسيلة، يتم حفظها.

٢. إجراء التحديث بالصف.

٣. استدعاء وسيلة EndEdit لتثبيت التغييرات بالصف وإلغاء تعليق القيود. ويترتب على ذلك وقوع حدث RowChanging.

٤. نستدعى وسيلة CancelEdit لإزالة التغييرات من الصف.

المثال التالى يوضح عملية تحديث بسيطة موضوعة بين عملية استدعاء وسيلة BeginEdit وبين عملية استدعاء وسيلة EndEdit. ويفترض المثال وجود جدول بيانات باسم Customers فى فئة بيانات DsCustomers1:

```
DsCustomers1.Customers (4).BeginEdit ()
DsCustomers1.Customers (4).CompanyName = "Wingtip Toys"
DsCustomers1.Customers (4).City = "Buffalo"
DsCustomers1.Customers (4).EndEdit ()
```

دمج فئات البيانات

يمكن دمج محتويات فئة بيانات يشار إليها بفئة بيانات المصدر فى فئة بيانات أخرى يشار إليها بفئة بيانات المقصد. من أمثلة دمج فئات البيانات ما يلى:

- قد يكون لدينا فئة بيانات رئيسية تحتوى على البيانات العاملة، ونحصل على فئة بيانات أخرى تحتوى على تحديث للبيانات أو سجلات جديدة مطلوب إضافتها إلى الفئة الأصلية.
- يقوم التطبيق بالحصول على فئات بيانات من مصادر متنوعة ولكنة يحتاج إلى العمل مع فئة بيانات واحدة. على سبيل المثال، يمكن الحصول على معلومات إحدى الشركات من أحد المصادر ومعلومات أسعار الأسهم من مصدر أخر.

لنسخ المعلومات من فئة بيانات إلى أخرى، نستخدم وسيلة Merge الخاصة بفئة

بيانات المقصد، ونمرر إليها فئة بيانات المصدر التى تحتوى على السجلات التى نريد دمجها.

```
targetDataset.Merge (sourceDataset)
```

تثبيت التغييرات فى فئة البيانات

عند إجراء تعديلات بالسجلات فى فئة البيانات عن طريق التحديث، الإدراج، أو الحذف، تحتفظ فئة البيانات بنسخة من السجل الأصلى ونسخة من السجل الجاري. بالإضافة إلى ذلك، يتم تغيير خاصية RowState الخاصة بكل صف للإشارة إلى أن السجلات فى حالتها الأصلية أو أن السجلات قد تم تغييرها، إدراجها، أو حذفها. وتظهر فائدة هذه المعلومات عندما نريد العثور على نسخة معينة من الصف. بعد ذلك تستخدم السجلات التى حدث بها تغيير والسجلات الجديدة فى تحديث مصدر البيانات الأصلى. بعد الانتهاء من تحديث مصدر البيانات بالتغييرات، يجب تغيير حالة السجلات فى فئة البيانات لتصبح سجلات غير متغيرة. يتم ذلك عن طريق تثبيت البيانات باستخدام وسيلة AcceptChanges. يترتب على تنفيذ هذه الوسيلة، تغيير خاصية RowState فى كل كائن من كائنات DataRow بفئة البيانات. تتغير الصفوف المضافة (Added Rows) والمتغيرة (Modified Rows) إلى صفوف غير متغيرة (UnChanged Rows) ويتم حذف الصفوف التى بها إشارة الحذف (Deleted Rows) من فئة البيانات. الكود التالى يبين استخدام وسيلة AcceptChanges بعد تحديث مصدر البيانات بالتغييرات التى حدثت بفئة البيانات:

```
OleDbDataAdapter1.Update (Dataset1, "Customers")
Dataset1.Customers.AcceptChanges ()
```

ونقوم أيضا بتثبيت التغيرات باستخدام وسيلة AcceptChanges بعد دمج فئات البيانات، كما يتضح من الكود التالى الذى يقوم بدمج فئتين من فئات البيانات ثم يستدعى وسيلة AcceptChanges لتثبيت التغييرات:

```
Dataset1.Merge (Dataset2)
Dataset1.AcceptChanges ()
```

التعرف على الصفوف المتغيرة واستخراجها

قبل استخدام سجلات فئات البيانات فى تحديث مصادر البيانات الأصلية، يجب التعرف على السجلات التى حدث بها تغيير فى فئة البيانات لكى يمكن استخدامها فى

عملية التحديث. كما يجب تحديد نوع التغير الذى حدث بالسجل لكى يتم تنفيذ عملية التحديث المناسبة. وللقيام بعملية التحديث بعد ذلك، يجب استخراج صفوف البيانات التى حدث بها التغيير تمهيدا لاستخدامها فى تحديث مصادر البيانات.

فحص الصفوف المتغيرة فى فئة البيانات

عند إحداث تغييرات فى سجلات فئات البيانات، يتم تخزين معلومات هذه التغييرات إلى أن يتم تثبيت هذه التغييرات. ويتم تعريف التغييرات التى حدثت فى الصفوف باستخدام طريقتين:

- كل سجل يحفظ نوع التغييرات التى حدثت به باستخدام خاصية RowState بكائن DataRow.
- تحتفظ فئات البيانات بأكثر من نسخة من الصفوف المتغيرة، النسخة الأصلية قبل التغيير والنسخة الجارية بعد التغيير. كما توجد نسخة ثالثة أثناء تعليق التغيير وهى النسخة المقترحة (Proposed Version).

تحديد الصفوف المتغيرة

لتحديد ما إذا كانت هناك تغييرات قد حدثت فى أحد الصفوف، نستدعى وسيلة HasChanges فى فئة بيانات لفحص الصفوف المتغيرة. يوضح هيكل الكود التالى كيفية فحص القيمة العائدة من هذه الوسيلة لاكتشاف ما إذا كانت هناك صفوف قد تم تغييرها فى فئة بيانات باسم myDataset:

```
If myDataset.HasChanges () Then

Else

End If
```

تحديد نوع التغييرات التى حدثت بالصف

يمكن فحص نوع التغييرات التى حدثت فى فئة بيانات عن طريق تمرير قيمة من قيم تعداد DataRowState إلى وسيلة HasChanges. يوضح هيكل الكود التالى كيفية فحص فئة بيانات تسمى myDataset لمعرفة ما إذا كانت هناك صفوف قد تم إضافتها:

```
If myDataset.HasChanges (DataRowState.Added) Then

Else
  End If
```

استخراج الصفوف المتغيرة

نحتاج إلى استخراج الصفوف المتغيرة من فئة بيانات لإرسالها إلى مصدر البيانات أو برنامج أخر. ويتم تحقيق ذلك باستخدام وسيلة GetChanges فى فئة أو جدول بيانات. تعيد هذه الوسيلة فئة بيانات جديدة أو جدول بيانات يحتوى على السجلات التى حدث بها تغيير فقط. وعندما نريد الحصول على نوع معين من السجلات، مثل السجلات الجديدة أو المتغيرة، نقوم بتمرير معامل خاص بحالة الصف إلى هذه الوسيلة.

المثال التالى يقوم بتكوين فئة بيانات جديدة تسمى myChangedRecords وتأهيلها بكل السجلات التى حدث بها تغيير فى فئة بيانات أخرى تسمى myDataset:

```
Dim myChangedRecords as DataSet
myChangedRecords = myDataSet.GetChanges ()
```

ويمكن الحصول على السجلات التى تغيرت وتحتوى على خاصية RowState محددة عن طريق استخدام وسيلة GetChanges فى فئة أو جدول بيانات وتمرير قيمة DataRowState إليها. المثال التالى يوضح كيفية تكوين فئة بيانات جديدة باسم myChangedRecords وتأهيلها بالسجلات المضافة فقط إلى فئة بيانات myDataset:

```
Dim myChangedRecords as DataSet
myChangedRecords = myDataSet.GetChanges (DataRowState.Added)
```

البحث عن سجل محدد فى فئة البيانات

تحتاج التطبيقات فى الغالب إلى الوصول إلى سجلات محددة تتوافق مع أحد المعايير. للعثور على صف محدد فى فئة البيانات، نستخدم وسيلة Find فى كائن DataRowCollection. عند تواجد المفتاح الرئيسى، تعيد هذه الوسيلة كائن DataRow. وتعيد القيمة null عندما لا تجد المفتاح الأساسى. المثال التالى يوضح كيفية الإعلان عن صف جديد وتخصيص القيمة العائدة من وسيلة Find له. وعند العثور على المفتاح الأساسى، يتم عرض محتويات العمود الأول فى الفهرس فى مربع رسالة:

```
Dim n As String = "primaryKeyValue to be found"
Dim rowFoundRow as DataRow = anyTable.Rows.Find (n)
If Not (rowFoundRow Is Nothing) Then
  MessageBox.Show (CType (rowFoundRow (1), String))
Else
  MessageBox.Show ("A row with the primary key of " & _
    n & " could not be found")
End If
```

الحصول على نسخة محددة من أحد الصفوف

تتواجد النسخ المختلفة من الصفوف بعد تدقيق هذه الصفوف وقبل استدعاء وسيلة AcceptChanges. وبعد استدعاء وسيلة AcceptChanges، تصبح النسخة الأصلية والنسخة الجارية نفس الشيء. للحصول على نسخة محددة من أحد الصفوف، نقوم بالوصول إلى أحد حقول الصف ونضيف معامل إلى الفهرس يشير إلى نسخة الصف الذى نريد الحصول علية. المثال التالى يوضح كيفية استخدام قيمة DataRowVersion للحصول على القيمة الأصلية فى عمود CompanyName:

```
Dim sOrigCompName As String
sOrigCompName = CType (DsCustomers1.Customers (0)("CompanyName", _
DataRowVersion.Original), String)
```

الوصول إلى الصفوف التى بها أخطاء

إذا لم يستخدم التطبيق إجراءات معالجة لأحداث RowUpdated أو ColumnUpdated لاصطياد الأخطاء أثناء عملية التحديث، سوف نحتاج إلى العثور على الخطأ باستخدام الكود. لتحديد صف به خطأ، نقوم بضبط خاصية HasErrors بكائن الصف، ثم نقوم بتكرار القراءة خلال مجموعة الجداول ثم مجموعة الصفوف للعثور على الصف الذى به خطأ.

```
Private Sub finderr2 ()
   Dim dt As DataTable
   Dim dr As DataRow
   For Each dt In anyDataset.Tables
       For Each dr In dt.Rows
          If dr.HasErrors Then
             dr.ClearErrors ()
          End If
        Next
   Next
```

```
End Sub
```

تدقيق صحة البيانات فى فئات البيانات

عند تغيير القيم فى صفوف الجداول بفئات البيانات، تقع أحداث معينة يمكن استخدامها فى تكوين إجراءات معالجة تقوم بتدقيق تغييرات البيانات. ويمكن أن تقوم فئات البيانات بعمليات التدقيق بغض النظر عن الطريقة التى يتم بها التحديث، سواء كان التحديث مباشر عن طريق أدوات التحكم فى نموذج أو بطريقة أخرى. ويمكن إنجاز عمليات المراجعة داخل فئة البيانات باتباع الوسائل التالية:

- عن طريق إنشاء مفاتيح، قيود تفرد، وغيرها باعتبارها جزءا من تعريف المخطط الخاص بفئة البيانات.
- عن طريق إنشاء إجراءات التدقيق التى تقوم باختبار البيانات أثناء تغيير الأعمدة والصفوف.
- عن طريق ضبط خصائص كائن DataColumn، مثل خاصية AllowDBNull، MaxLength، وخاصية Unique.

ويمكن تمييز مجموعتان من الأحداث التى تقع بواسطة كائن DataTable عند حدوث تغيير فى سجل:

- أحداث تتعلق بتغيير أعمدة البيانات فى الجداول. تشمل حدث ColumnChanging الذى يقع أثناء تغيير بيانات الأعمدة، وحدث ColumnChanged الذى يقع بعد الانتهاء من تغيير بيانات العمود. ويعتبر حدث ColumnChanging مفيدا فى تدقيق التغييرات قبل تنفيذها. ويتم تمرير التغييرات المقترحة فى صورة معامل لهذا الحدث.
- أحداث تتعلق بالتغيير فى صفوف البيانات بالجداول. تشمل حدث RowChanging، وحدث RowChanged. تقع حدث RowChanging أثناء التغيير فى أحد الصفوف، بينما يقع حدث RowChanged بعد تغيير بيانات الصف. ويعتبر استخدام حدث RowChanging أكثر شيوعا لأنة يشير إلى أن هناك تغييرا يحدث فى مكان ما بالصف بدون تحديد العمود الذى يتغير.

يترتب على التغيير فى عمود وقوع أربعة أحداث: حدث ColumnChanging، حدث ColumnChanged، حدث RowChanging، وحدث RowChanged. ويعتمد اختيارنا للحدث الذى نقوم بمعالجته على طريقة المراجعة المناسبة لنا. إذا كان من المهم تصعيد الأخطاء مباشرة عند تغيير عمود، نقوم ببناء إجراء التدقيق باستخدام حدث ColumnChanging. وعندما يقوم هيكل المراجعة على أساس مراجعة صحة قيمة عمود على أساس محتويات عمود أخر، يجب استخدام حدث RowChanging لمراجعة صحة البيانات.

مراجعة البيانات أثناء تغيير عمود

يمكن مراجعة البيانات عندما يتغير أحد الأعمدة عن طريق الاستجابة لحدث ColumnChanging. يقوم هذا الحدث بتمرير كائن ذات خصائص تسمح لنا بالوصول إلى العمود الذى يتغير. للقيام بهذا النوع من المراجعة، ننفذ الخطوات التالية:

١. نقوم بتكوين معالج حدث ColumnChanging.

٢. فى معالج الحدث، يمكننا الحصول على القيم المقترحة وعلى القيم الأصلية عن طريق فحص خاصية ProposedValue وخاصية Row، كما يتضح من الكود التالى:

```
Dim newValue As String = CType (e.ProposedValue, String)
Dim origvalue As String = CType (e.Row (e.Column), String)
```

٣. للحصول على العمود الذى يحدث به التغيير، نقوم بفحص خاصية Column التى يتم تمريرها إلى إجراء المعالجة، كما يتضح من الكود التالى:

```
Dim colDataType As String = e.Column.DataType.ToString ()
Dim colName As String = e.Column.ColumnName
```

٤. لرفض التغيير، نقوم بإطلاق خطأ استثنائي:

```
Private Sub dt_ColumnChanging (ByVal sender As Object, ByVal e As
System.Data.DataColumnChangeEventArgs) Handles dt.ColumnChanging
  Dim newvalue As Integer = CType (e.ProposedValue, Integer)
  If newvalue < 10 Then
    MessageBox.Show (newvalue.ToString () & " is not less than 10")
    Throw New Exception (newvalue.ToString () & " is not less than 10")
  End If
End Sub
```

مراجعة البيانات أثناء تغيير الصف

يمكن مراجعة صحة البيانات عند تغيير الصف عن طريق الاستجابة لحدث RowChanging الذى يقع فى تصنيفات فئات البيانات النوعية. على سبيل المثال، عندما تحتوى فئة بيانات نوعية على جدول باسم Empolyees، يقوم كائن هذا الجدول بإطلاق حدث EmployeesRowChanging. ويقوم هذا الحدث بتمرير كائن يحتوى على خصائص تسمح لنا بالوصول إلى العمود الذى يجرى تغييره.

للقيام بمراجعة البيانات أثناء تغيير بيانات أحد الصفوف، ننفذ الخطوات التالية:

١. نقوم بإنشاء معالج لحدث RowChanging الموجود فى جدول البيانات.

٢. فى إجراء المراجعة، نحصل على التغييرات التى حدثت بالصف عن طريق استخراج الأعمدة من النسخة المقترحة (Proposed Version)، باستخدام الكود التالى:

```
newFName = CType (e.Row ("FirstName", DataRowVersion.Proposed), String)
```

ومن الملاحظ أن نسخة Proposed تكون متاحة فى صورة خصائص نوعية للصف عند العمل مع حدث خاص بالجدول ويمكن الوصول إليها باستخدام الكود التالى:

```
newFName = e. Row. FirstName.
```

٣. نحصل على الإصدارات الموجودة باستخدام الكود التالى:

```
oldFName = CType (e.Row ("FirstName", DataRowVersion.Original), String)
```

٤. إنجاز عملية المراجعة. لرفض التغيير، نطلق خطأ استثنائي. ولقبول التغيير ، لا نحتاج إلى كتابة كود.

يوضح المثال التالى كيفية مراجعة بيانات فى حدث RowChanging. فى هذا المثال، يقوم الكود باختبار وجود قيمة فى عمود SecurityCode. وعند وجود قيمة لايجب أن تكون هذه القيمة null. ويقوم الكود بالتحقق من وجود قيمة باستخدام خاصية HasVersion بصف البيانات.

```
Private Sub employeetable_EmployeesRowChanging (ByVal sender As _
```

```
System.Object, ByVal e As dsEmployees.EmployeesRowChangeEvent) _
Handles employeetable.EmployeesRowChanging
  Dim original As String = ""
  Dim proposed As String = ""
  If e.Row.HasVersion (DataRowVersion.Original) Then
    original = CType (e.Row ("SecurityCode", DataRowVersion.Original), String)
  Else
    original = ""
  End If
  proposed = e.Row.SecurityCode
  If original <> "" Then
    If proposed = "" Then
      Throw (New Exception ("Security Code cannot be blank"))
    End If
  End If
End Sub
```

تحديث مصادر البيانات

بعد تعديل محتويات فئة بيانات ومراجعة هذه التعديلات، سوف تحتاج معظم التطبيقات إلى كتابة التغييرات فى قاعدة البيانات. الوسيلة المستخدمة فى تنفيذ هذه العملية هى وسيلة Update الموجودة فى موفق البيانات. تقوم هذه الوسيلة باستخدام حلقة لتكرار قراءة السجلات فى جدول البيانات، كما تقوم بتحديد نوع التحديث المطلوب وبالتالى تنفيذ الأمر المناسب.

تحويل التغييرات إلى مصدر البيانات

لتوضيح كيفية إجراء عملية التحديث، نفترض وجود تطبيق يستخدم فئة بيانات تحتوى على جدول بيانات وحيد. يقوم التطبيق باستخراج صفين من قاعدة البيانات ووضعهما فى جدول فئة البيانات، الذى سوف يبدو بالشكل التالى:

(RowState)	CustomerID	Name	Status
(Unchanged)	c200	Robert Lyon	Good
(Unchanged)	c400	Nancy Buchanan	Pending

يقوم التطبيق بتغيير قيمة حقل Status فى السجل الثانى إلى "Preferred". نتيجة لهذا التغيير، تتغير خاصية DataRow.RowState من Unchanged إلى Modified. وتبقى

قيمة هذه الخاصية بالنسبة للصف الأول بدون تغيير. سوف يبدو جدول البيانات بعد ذلك على الشكل التالى:

(RowState)	CustomerID	Name	Status
(Unchanged)	c200	Robert Lyon	Good
(Modified)	c400	Nancy Buchanan	Preferred

يقوم التطبيق بعد ذلك باستدعاء وسيلة Update لإرسال التغييرات إلى قاعدة البيانات. تقوم هذه الوسيلة بفحص كل صف من الصفوف. بالنسبة للصف الأول، لا يتم إرسال عبارات SQL إلى قاعدة البيانات، لأن هذا الصف لم يتغير منذ الحصول علية من قاعدة البيانات. وبالنسبة للصف الثانى، تقوم وسيلة Update تلقائيا بإرسال الأمر المناسب إلى قاعدة البيانات. وتختلف عبارة SQL فى أمر البيانات المرسل على أساس نوع التحديث المطلوب.

فى هذا المثال، سوف تكون عبارة SQL هى عبارة UPDATE. وسوف تشتمل هذه العبارة على فقرة WHERE التى تحدد المقصد الذى يجب تحديثه. فى هذا المثال، سوف تحتوى فقرة WHERE على تعبير (CustomerID = 'c400')، حيث يمثل عمود CustomerID المفتاح الأساسى فى الجدول المستهدف بقاعدة البيانات. ويتم استخراج المعلومات الخاصة بفقرة WHERE من النسخة الأصلية للسجل بفئة البيانات.

تمرير المعاملات

من المعتاد تمرير القيم الخاصة بالسجلات التى يجرى تغييرها فى قاعدة البيانات باستخدام المعاملات (Parameters). وعندما تقوم وسيلة Update بتنفيذ عبارة UPDATE، فإنها تحتاج إلى الحصول على قيم لهذة المعاملات. يتم الحصول على هذه القيم من مجموعة المعاملات الخاصة بأمر البيانات (Data Command) الموجود فى موفق البيانات. وعند استخدام أدوات Visual Studio للحصول على موفق بيانات، سوف يحتوى كائن UpdateCommand على مجموعة من المعاملات التى تقابل كل موقع من مواقع مجموعة المعاملات فى العبارة المستخدمة لتحديث قاعدة البيانات.

بالنسبة لتصنيف OleDbDataAdapter، يتم التعرف على المعاملات بالموقع. بمعنى أن الموقع الأول فى العبارة سوف يحتوى على قيمة المعامل الأول فى المجموعة. وبالنسبة

لتصنيف SqlDataAdapter، يتم التعرف على المعاملات بالاسم.

إجراءات تحديث قواعد البيانات باستخدام فئات البيانات

قد يختلف الإجراء المستخدم فى تحديث قواعد البيانات، غير أن هناك خطوات مشتركة يجب أن يشتمل عليها التطبيق المستخدم :

١. استدعاء وسيلة Update فى موفقات البيانات داخل مجمع Try...Catch.
٢. عند اصطياد أحد الأخطاء، نحدد صف البيانات الذى تسبب فى الخطأ.
٣. تصحيح الخطأ عن طريق الكود أو بعرض الخطأ أمام المستخدم لكى يقوم بتصحيحه، ثم محاولة التحديث من جديد.

المثال التالى يبين كيفية تحديث مصدر بيانات من داخل مجمع Try...Catch باستخدام محتويات فئة بيانات باسم myDataset:

```
Try
   OleDbDataAdapter1.Update (myDataset)
Catch x As Exception
        نضيف الكود اللازم لتحديد موقع الأخطاء التى قد تحدث وطريقة معالجتها
        نقوم بإعادة محاولة التحديث
End Try
```

تحديث الجداول ذات العلاقات فى قاعدة البيانات

عندما تحتوى فئة البيانات على جداول متعددة، يجب تحديث الجداول المقابلة فى مصدر البيانات، كل على انفراد عن طريق استدعاء وسيلة Update فى موفقات البيانات الخاصة بهذه الجداول. وعندما تكون هناك جداول أصلية وجداول تابعة، يكون من الملائم إرسال التحديث إلى قاعدة البيانات فى تسلسل محدد. القاعدة العامة التى تحكم إرسال التحديث الخاص بالجداول التى بينها علاقات هى اتباع التسلسل التالى:

١. إرسال الجدول التابع : حذف سجلات.
٢. إرسال الجدول الأصلى: إدراج، تحديث، وحذف سجلات.
٣. إرسال الجدول التابع : إدراج وتحديث سجلات.

المثال التالى يوضح كيفية تحديث مصدر بيانات باستخدام فئة بيانات تحتوى على جداول ذات علاقات. يحتوى المثال على ثلاثة فئات بيانات مؤقتة لاستخدامها فى الاحتفاظ بالسجلات المختلفة. يتم بعد ذلك استدعاء وسيلة Update لكل مجموعة فرعية من السجلات من داخل مجمع Try...Catch. عند حدوث أخطاء فى التحديث، يتم تنفيذ التصرفات الخاصة بإصلاح الخطأ ثم إعادة محاولة التحديث. بعد الانتهاء من التحديث، يتم تثبيت التغييرات فى فئة البيانات والتخلص من فئات البيانات المؤقتة.

```
Private Sub Update_Attempt ()
```

تكوين جدول البيانات المؤقت ds1 لكى يحتوى على السجلات المحذوفة من الجدول التابع :

```
Dim ds1 as DataTable = _
    anyDataset.ChildTableName.GetChanges (DataRowState.Deleted)
```

تكوين جدول البيانات المؤقت ds2 لكى يحتوى على السجلات الجديدة من الجدول التابع :

```
Dim ds2 as DataTable = _
    anyDataset.ChildTableName.GetChanges (DataRowState.Added)
```

تكوين جدول البيانات المؤقت ds3 لكى يحتوى على الصفوف المتغيرة من الجدول التابع :

```
  Dim ds3 as DataTable = _
    anyDataset.ChildTableName.GetChanges (DataRowState.Modified)

  Try
    DataAdapter2.Update (ds1)
    DataAdapter1.Update(anyDataset, "ParentTable")
    DataAdapter2.Update(ds2)
    DataAdapter2.Update(ds3)
    anyDataset.AcceptChanges
    ds1.Dispose()
    ds2.Dispose()
    ds3.Dispose()
  Catch x As Exception
```

نضيف الكود اللازم لتحديد موقع الخطأ وطريقة معالجته.

نحاول التحديث مرة أخرى بعد تصحيح الخطأ.

```
  End Try
```

End Sub

الاستجابة لأخطأ تحديث قواعد البيانات

من الممكن أن تقع أخطاء عند محاولة تحديث قاعدة البيانات. من الأخطاء الشائعة تغير قاعدة البيانات بعد قراءتها فى فئة البيانات. وبغض النظر عن سبب الخطأ، يجب كتابة الكود اللازم لتصحيح السجل الذى وقع به. أول خطوات تصحيح الخطأ هو العثور على الصف الذى حدث به هذا الخطأ. يلى ذلك حل المشكلة التى أحدثت الخطأ والتخلص منة ثم محاولة التحديث مرة أخرى.

عندما يتم تنفيذ أمر تحديث بواسطة موفق بيانات (Data Adapter)، يقع حدث RowUpdated لكل سجل يتأثر بهذا الأمر. عن طريق كتابة إجراء معالجة لهذا الحدث، يمكن الحكم على نجاح عملية التحديث. ويمكن أيضا ضبط خاصية Status لتوجيه أمر التحديث بالاستمرار فى معالجة التغيرات التالية أو إيقاف التحديث بعد حدوث الخطأ. للاستجابة لخطأ تحديث قاعدة بيانات، نتبع الخطوات التالية:

١. نكون إجراء معالجة لحدث RowUpdated.

٢. نفحص خاصية Status داخل إجراء معالجة حدث RowUpdated. سوف تحتوى هذه الخاصية على ErrorsOccured عند حدوث خطأ.

٣. نضبط خاصية Status على القيمة المناسبة فى تعداد UpdateStatus، كما فى الجدول رقم (٢٥).

القيمة	الإيضاح
Continue	يستمر موفق البيانات فى معالجة الصفوف
ErrorsOccurred	اعتبار أن هناك خطأ فى عملية التحديث
SkipAllRemainingRows	عدم تحديث الصف الحالى وكل الصفوف التالية
SkipCurrentRow	عدم تحديث الصف الحالى

جدول ٢٥

يوضح المثال التالى كيفية اختبار نجاح تحديث أحد الصفوف داخل إجراء معالجة حدث RowUpdated. وإذا كان هناك خطأ فى عملية التحديث، يتم ضبط خاصية RowError فى صف البيانات ذات العلاقة على رسالة الخطأ، التى بدورها تضبط خاصية HasErrors على القيمة True. عند هذه النقطة، يمكن معالجة الخطأ بناءا على نوع الخطأ الذى تم رفعة، وبناءا على احتياجات التطبيق. بالإضافة إلى ذلك، يمكن ضبط خاصية Status للاستمرار فى معالجة الصفوف، هجر تحديث باقى الصفوف، أو هجر تحديث الصف الحالى.

```
Private Sub OleDbDataAdapter1_RowUpdated (ByVal sender As Object, _
ByVal e As System.Data.OleDb.OleDbRowUpdatedEventArgs) _
Handles OleDbDataAdapter1.RowUpdated
  If e.Status = UpdateStatus.ErrorsOccurred Then
    e.Row.RowError = e.Errors.Message
```

نضيف الكود اللازم لمعالجة الخطأ فى هذا الموقع.

```
  End If
End Sub
```

تجديد فئة البيانات

من المناسب فى بعض الظروف بعد الانتهاء من تحديث مصدر البيانات، تجديد (Refreshing) فئة البيانات عن طريق إعادة تأهيلها. يترتب على القيام بذلك عدد من الفوائد:

- الحصول على التغييرات التى تحدث فى قاعدة البيانات بواسطة المستخدمين الآخرين.
- الحصول على القيم التى تقوم قاعدة البيانات بحسابها، مثل قيم أعمدة تعريف السجلات أو الأعمدة ذات القيم الافتراضية.
- تجديد قيمة Timestamp على السجلات فى فئة البيانات، عند الرغبة فى استخدام Timestamp فى التحكم فى تزامن البيانات.

يمكن تجديد فئة البيانات يدويا عن طريق استدعاء وسيلة Fill فى موفق البيانات بعد استدعاء وسيلة Update. وبدلا عن ذلك، يمكن إعداد موفق البيانات لكى يقوم بتنفيذ عملية

التجديد بطريقة أتوماتيكية عن طريق تنفيذ عبارة SQL SELECT أو إجراء مخزن (Stored Procedure) بعد الانتهاء من عملية التحديث.

التحكم فى التزامن

بالنظر إلى أن فئة البيانات تستخدم منفصلة عن قاعدة البيانات، لذا يجب الرقابة على تحقق التزامن بين سجلات قاعدة البيانات وبين سجلات فئة البيانات. على سبيل المثال، قد نجد أن السجلات فى قاعدة البيانات قد تغيرت منذ أخر تأهيل لفئة البيانات. فى هذه الحالة، يجب كتابة الكود المناسب لتحديد ما يجب عملة بالنسبة لسجل قاعدة البيانات أو السجل المقابل فى فئة البيانات.

أنواع التحكم فى التزامن

هناك ثلاثة أساليب شائعة لإدارة التزامن في قاعدة البيانات:

- أسلوب Pessimistic Concurrency Control. بموجب هذا الأسلوب لا يكون الصف متاحا لباقى المستخدمين بعد استخراجه من قاعدة البيانات إلى أن يتم تحديثه فيها بواسطة نفس المستخدم.
- أسلوب Optimistic Concurrency Control. لا يكون الصف متاحا لباقى المستخدمين أثناء عملية التحديث فقط. ويتم فحص الصف فى قاعدة البيانات لتقرير ما إذا كان الصف قد تم تعديله. ولا يسمح بتحديث الصف الذى تم تعديله بواسطة المستخدمين الآخرين.
- أسلوب Last in wins. بموجب هذا الأسلوب لا يكون الصف متاحا أمام باقى المستخدمين أثناء عملية التحديث فقط. كما لا يتم التحقق من تحديث الصف بواسطة المستخدمين الآخرين. ويتم كتابة أخر تحديث بالصف، مما يترتب علية الكتابة على التحديث السابق.

يستخدم كل من ADO.NET و Visual Studio.NET أسلوب Optimistic Concurrency بسبب استخدام هياكل البيانات غير المتصلة المتمثلة فى فئات البيانات. ولتنفيذ هذا الأسلوب، يمكن استخدام أحد مدخلين: مدخل رقم الإصدار (The Version Approach)، ومدخل حفظ جميع القيم (The Saving All Values Approach).

مدخل رقم الإصدار لتحقيق التزامن (The Version Approach)

فى هذا المدخل يجب أن يكون بالسجل المطلوب تحديثه عمود يحتوى على رقم الإصدار (Version Number) أو بصمة التاريخ والوقت (Date-time Stamp). ولتحقيق التزامن باستخدام بصمة الوقت، نقوم بالتحديث فقط عند تطابق القيمة التى من هذا النوع فى السجل المطلوب تحديثه والقيمة الموجودة فى فقرة WHERE بعبارة SQL UPDATE، كما يتضح من الكود التالى:

```
UPDATE Table1 SET Column1 = @newvalue1, Column2 = @newvalue2
WHERE DateTimeStamp = @origDateTimeStamp
```

ويمكن استخدام رقم الإصدار بدلا من بصمة الوقت، كما يتضح من الكود التالى:

```
UPDATE Table1 SET Column1 = @newvalue1, Column2 = @newvalue2
WHERE RowVersion = @origRowVersionValue
```

مدخل حفظ جميع القيم لتحقيق التزامن (The Saving All Values Approach).

فى هذا المدخل، يتم الحصول على نسخ من جميع الحقول عند قراءة السجل. وهو ما يحدث فى كائن DataSet، حيث يتم الاحتفاظ بالنسخة الأصلية من البيانات والنسخة المعدلة. وعند القيام بتحديث مصدر البيانات بالتغييرات الحادثة فى فئة البيانات، يتم مقارنة القيم الأصلية فى صف فئة البيانات مع السجل الأصلى فى مصدر البيانات. عند تطابق السجلين، تتم عملية التحديث بنجاح. المثال التالى يوضح نص تحديث جدول العملاء فى قاعدة بيانات باستخدام البيانات المقابلة فى فئة بيانات :

```
UPDATE Customers SET CustomerID = @currCustomerID, CompanyName =
    @currCompanyName, ContactName = @currContactName,
    ContactTitle = currContactTitle, Address = @currAddress, City = @currCity,
    PostalCode = @currPostalCode, Phone = @currPhone, Fax = @currFax
WHERE (CustomerID = @origCustomerID) AND (Address = @origAddress OR
    @origAddress IS NULL AND Address IS NULL) AND (City = @origCity OR
    @origCity IS NULL AND City IS NULL)
    AND (CompanyName = @origCompanyName OR @origCompanyName IS
    NULL AND CompanyName IS NULL) AND (ContactName =
    @origContactName OR @origContactName IS NULL AND ContactName IS
    NULL) AND (ContactTitle = @origContactTitle OR @origContactTitle IS
    NULL
    AND ContactTitle IS NULL)
```

```
     AND (Fax = @origFax OR @origFax IS NULL AND Fax IS NULL) AND
     (Phone = @origPhone OR @origPhone IS NULL AND Phone IS NULL) AND
     (PostalCode = @origPostalCode OR @origPostalCode IS NULL AND
     PostalCode IS NULL);

SELECT CustomerID, CompanyName, ContactName, ContactTitle, Address, City,
     PostalCode, Phone, Fax
FROM Customers WHERE (CustomerID = @currCustomerID)
```

يجب ملاحظة أن المعاملات التسعة لعبارة SET تمثل القيم الحالية التى سوف تكتب فى قاعدة البيانات، بينما تمثل المعاملات التسعة فى فقرة Where القيم الأصلية التى تستخدم فى تحديد السجل الأصلي. وتستخدم عبارة SELECT المذكورة فى أخر النص لتجديد فئة البيانات بعد الانتهاء من عملية تحديث قاعدة البيانات. ويتم تنفيذ هذه العبارة عند ضبط خيار Refresh the DataSet فى مربع Advanced SQL Generations Options.

تحقيق التزامن باستخدام Dynamic SQL

يمكن أن يقوم Visual Studio بتنفيذ مدخل Version Approach أو مدخل Saving All Values الخاص بتحقيق التزامن، من خلال استخدام Dynamic SQL أو من خلال استخدام Stored Procedures. لبناء عبارات SQL التى تحقق التزامن بين البيانات الموجودة فى فئات البيانات وبين تلك الموجودة فى قواعد البيانات، نطبق الخطوات التالية:

١. نسحب جدول (Table) من مربع Server Explorer إلى المصمم. يترتب على ذلك تكوين كائن DataAdapter وكائن Connection فى مربع المكونات أسفل مصمم النماذج.

٢. ننقر بزر الماوس الأيمن على مكون DataAdapter ثم نختار Configure Data Adapter لعرض مربع حوار DataAdapter Configuration Wizard.

٣. ننقر Next ثم ننقر Next مرة أخرى لعرض صفحة Choose a Query Type.

٤. نتأكد من اختيار Use SQL Statements، ثم ننقر Next.

٥. فى صفحة Generate the SQL Statements ثم ننقر زر Query Builder.

٦. فى مربع حوار Query Builder، نختار الحقول التى نريدها.

٧. بعد اختيار الأعمدة المطلوبة ، ننقر Ok.

٨. فى صفحة Generate the SQL Statements، ننقر زر Advanced Options مع ملاحظة اختيار Use Optimistic Concurrency تلقائيا.

٩. ننقر Ok ثم Finish للانتهاء من العملية.

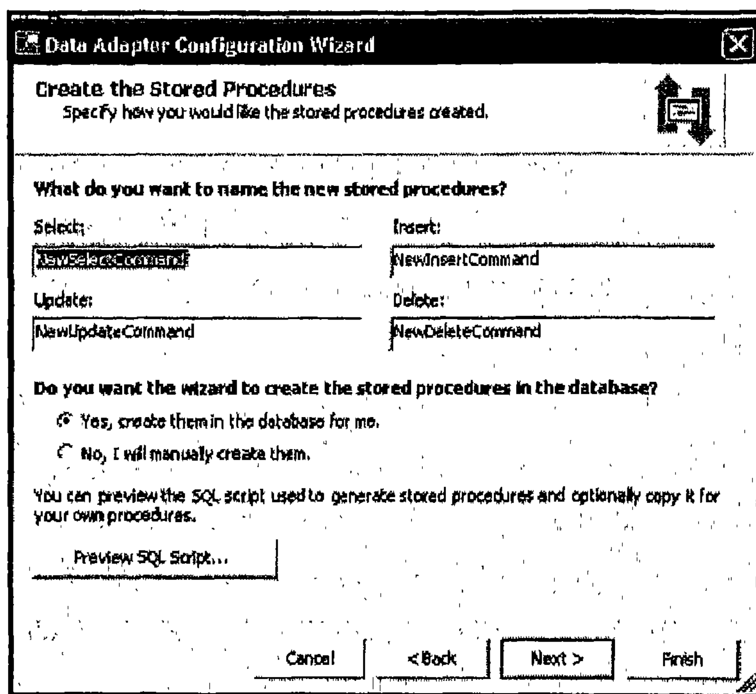
يمكن فحص نتائج عملية الإعداد عن طريق فحص خصائص أوامر موفق البيانات التى تشمل DeleteCommand ، InsertCommand ، UpdateCommand.

تحقيق التزامن باستخدام Stored Procedures

يمكن أن يقوم Visual Studio بتنفيذ Optimistic Concurrency باستخدام الإجراءات المخزنة عن طريق بناء أمر SQL يشتمل على فقرة WHERE التى تحتوى على كل القيم الأصلية. ويتم تمرير القيم الموجودة فى فقرة WHERE إلى إجراء مخزن داخل مصدر البيانات. وسوف لا يترتب على تنفيذ الأمر إعادة أى سجلات عند تغير بيانات المصدر. لبناء أمر SQL المذكور، نطبق الخطوات التالية:

١. نسحب جدول بيانات من مربع Server Explorer إلى مصمم النماذج. يترتب على ذلك تكوين كائن DataAdapter وكائن Connection فى مربع المكونات أسفل مصمم النماذج.

٢. ننقر بزر الماوس الأيمن على كائن DataAdapter، ثم نختار Configure Data Adapter لعرض مربع حوار DataAdapter Configuration Wizard المبين بالشكل رقم (١٨).



شكل رقم ١٨

٣. ننقر Next ثم ننقر Next مرة أخرى لعرض صفحة Choose a Query Type.

٤. نتأكد من اختيار Create new stored procedures، ثم ننقر Next.

٥. في صفحة Generate the stored procedures، ننقر زر Query Builder.

٦. في مربع Query Builder، نختار الحقول المطلوبة ثم ننقر Ok.

٧. في صفحة Generate the stored procedures، ننقر زر Advanced Options ونلاحظ اختيار Use optimistic concurrency تلقائيا.

٨. ننقر Ok ثم ننقر Next مرة أخرى لعرض صفحة Create the stored procedures المبينة في الشكل رقم ( ).

٩. في صفحة Create the stored procedures، نخصص أسماء لكل من الإجراءات الأربعة التي سوف يتم تكوينها.

١٠. ننقر Next ثم Finish للانتهاء من العملية.

يمكن فحص نتائج الإعداد عن طريق فحص خصائص أوامر موفق البيانات التى تشمل DeleteCommand، InsertCommand، و UpdateCommand.

التعامل مع أخطاء التزامن

تقدم ADO.NET كائن DBConcurrencyExecption للمساعدة فى حل المشاكل الناشئة عن مخالفات التزامن. ويعيد كائن DBConcurrencyExecption صف البيانات الذى تسبب فى حدوث خطأ التزامن، كما يعيد أيضا الرسالة المرتبطة بهذا الخطأ.

المثال التالى يحاول تحديث بيانات أحد المصادر باستخدام محتويات فئة بيانات تسمى myDataset من داخل مجمع Try...Catch. وعند صدور أحد الأخطاء، يجرى عرض رسالة الخطأ مع الحقل الأول فى صف البيانات المسبب للخطأ. ويفترض هذا الكود وجود اتصال مع قاعدة البيانات، وجود فئة بيانات اسم myDataset، وأن أمر التحديث سوف يرفع مخالفة تزامن. ويطبق هذا المثال الخطوات التالية:

١. تنفيذ أمر تحديث قاعدة بيانات من داخل مجمع Try...Catch.

٢. عند وجود اعتراض، يتم فحص الصف والرسالة فى عبارة Catch لتقرير سبب المخالفة.

٣. تنفيذ الكود التالى لإزالة الخطأ، ويمكن تغييره بناءا على قواعد النشاط التجاري المعمول بها.

```
Try
  SqlDataAdapter1.Update (myDataset)
Catch ex As DBConcurrencyException
  Dim customErrorMessage As String
  customErrorMessage = ex.Message & vbCrLf
  customErrorMessage += Ctype (ex.Row.Item (0), String)
  MessageBox.Show (customErrorMessage)
End Try
```

العمليات فى ADO.NET

العمليات (Transactions) هى مجموعات من أوامر قاعدة البيانات التى يجب تنفيذها

حزمة واحدة. وتتكون العملية من سلسلة من عبارات SQL SELECT، SQL INSERT، SQL UPDATE، SQL DELETE. ويوفر استخدام العمليات للتطبيق، القدرة على حذف جميع التغييرات التى تم تنفيذها من داخل العملية عند حدوث خطأ فى أى جزء منها. على سبيل المثال، نفترض أن هناك تطبيق يحتوى على جدولين، أحدهما يمثل المخزون والأخر يمثل أوامر البيع. عند إضافة أمر بيع إلى جدول الأوامر، تحتاج الكميات المطلوبة إلى الخصم من رصيد الكمية فى جدول المخزون. وإذا نجح تحديث جدول الأوامر بينما فشل تحديث جدول المخزون، فإن تكامل البيانات سوف يتعرض للخطر. ولكى نضمن تحديث كلا الجدولين بنجاح، يمكن وضع أوامر التحديث الخاصة بالجدولين فى حزمة واحدة أو بمعنى أخر وضعها فى عملية. فإذا تم تحديث أحد الجدولين بنجاح بينما لم ينجح تحديث الأخر، يمكننا حذف كامل العملية وإزالة السبب الذى أدى إلى المشكلة ثم محاولة التحديث من جديد. ويجب ملاحظة أن قاعدة البيانات التى نعمل معها يجب أن تدعم استخدام العمليات لكى نستطيع استخدامها.

وهناك ثلاث أوامر أساسية خاصة بالعمليات: BEGIN، COMMIT، و ROLLBACK. تقوم عبارة BEGIN ببدء العملية، وتعتبر كل العبارات التى تلى عبارة BEGIN جزءا من هذه العملية. وتكتمل العملية باستخدام عبارة COMMIT، أو تلغى باستخدام عبارة ROLLBACK. المثال التالى يوضح استخدام لغة Transact-SQL فى SQL Server لتكوين عملية:

```
BEGIN TRAN
INSERT INTO account (account,amount,debitcredit) values (100,100,'d')
INSERT INTO account (account,amount,debitcredit) values (300,100,'c')
IF (@@ERROR > 0)
 ROLLBACK
ELSE
 COMMIT
```

فى ADO.NET، يتم التحكم فى العمليات باستخدام كائن Connection وكائن Transaction. لتنفيذ أحد العمليات فى ADO.NET، نطبق الخطوات التالية:

١. نستدعى وسيلة BeginTransaction الموجودة فى كائن Connection لبدء العملية. ويترتب على تنفيذ أمر BeginTransaction إعادة مرجع إلى العملية.

٢. نخصص كائن العملية لخاصية Transaction فى أمر البيانات الذى نريد تنفيذه.

٣. ننفذ الأوامر المطلوبة.

٤. نستدعى وسيلة Commit فى كائن العملية لاستكمال العملية ، أو وسيلة RollBack لإلغاء العملية.

المثال التالى يوضح استخدام عمليات SQL Server فى ADO.NET:

- تكوين وفتح اتصال مع قاعدة بيانات Northwind فى SQL Server.

```
Dim myConnection As SqlConnection = New SqlConnection ("Data Source=localhost;Initial Catalog=Northwind;Integrated Security=SSPI;")
myConnection.Open ()
```

- تكوين كائن عملية.

```
Dim myTrans As SqlTransaction = myConnection.BeginTransaction ()
```

- تكوين كائن أمر بيانات وربطة مع كائن الاتصال وكائن العملية.

```
Dim myCommand As SqlCommand = New SqlCommand
myCommand.Connection = myConnection
myCommand.Transaction = myTrans
Try
```

- محاول إدراج سجلات بجدول Region فى قاعدة بيانات Northwind.

```
 myCommand.CommandText = "Insert into Region (RegionID, RegionDescription)
  VALUES (100, 'Description')"
  myCommand.ExecuteNonQuery ()
  myCommand.CommandText = "Insert into Region (RegionID, RegionDescription)
  VALUES (101, 'Description')"
  myCommand.ExecuteNonQuery ()
```

- فى حالة نجاح العملية ، يتم كتابة سطر على الشاشة يفيد ذلك.

```
  myTrans.Commit ()
  Console.WriteLine ("Both records are written to database.")
Catch e As Exception
```

- فى حالة عدم نجاح العملية ، يتم كتابة سطر على الشاشة يفيد ذلك.

```
  myTrans.Rollback ()
  Console.WriteLine (e.ToString ())
  Console.WriteLine ("Neither record was written to database.")
Finally
```

- إقفال الاتصال.

```
  myConnection.Close ()
End Try
```

تطبيق على استخدام ADO.NET

التطبيق التالى يستخدم الإجراءات المخزنة فى الاتصال مع قاعدة بيانات Northwind فى SQL Server. ويستخدم هذا التطبيق جدول Categories للحصول على مجموعات المنتجات، كما يستخدم جدول Products للحصول على تفصيلات هذه المنتجات. ويفترض التطبيق تواجد العناصر التالية:

- قائمة تحتوى على بند mnuFile الذى يحتوى بدورة على بند mnuExit.
- متحكم TabControl يحتوى على ثلاثة صفحات: صفحة الاتصال مع خادم SQL Server، صفحة تعرض نتيجة تنفيذ إجراء مخزن بدون معاملات، وصفحة تحتوى على نتائج تنفيذ إجراء مخزن باستخدام أحد المعاملات.
- تحتوى صفحة الاتصال على متحكم Button لاستدعاء إجراء تكوين الإجراءات المخزنة.
- تحتوى الصفحة الثانية على متحكم Button لاستدعاء إجراء الحصول على أغلى عشرة منتجات. كما تحتوى على متحكم TextBox متعدد السطور لعرض النتائج.
- تحتوى الصفحة الثالثة على متحكم Button لاستدعاء إجراء الحصول على المنتجات التى تخص إحدى المجموعات، متحكم ComboBox لإستخدامة فى اختيار المجموعة، ومتحكم DataGrid لعرض النتائج الخاصة بالاستعلام.
- مجموعة من أدوات تحكم Label لعرض المعلومات المختلفة.

نعرض فيما يلى خطوات تكوين هذا التطبيق مع الإيضاحات اللازمة:

١. إضافة مراجع إلى مناطق الأسماء (Namespaces) التى تحتوى على الكائنات المستخدمة.

```
Option Strict On
Imports System.Data.SqlClient
```

```
Imports System.Text

Public Class frmMain
   Inherits System.Windows.Forms.Form
```

٢. تكون الثوابت (Constants) التي تحتوى على سلاسل الاتصال التى نحتاج إليها، وعلى الثوابت التى تتحكم فى عرض رسائل الأخطاء.

```
   Protected Const SQL_CONNECTION_STRING As String = _
      "Server=localhost;" & _
      "DataBase=Northwind;" & _
      "Integrated Security=SSPI"
   Protected DidPreviouslyConnect As Boolean = False
   Protected HasCreatedSprocs As Boolean = False
   Protected strConn As String
```

٣. إجراء معالجة النقر على زر تكوين الإجراءات المخزنة.

```
      Private Sub btnCreateSprocs_Click(ByVal sender As System.Object, ByVal e
As
      System.EventArgs) Handles btnCreateSprocs.Click

      strConn = SQL_CONNECTION_STRING
```

٤. عرض رسالة تفيد محاولة الاتصال مع قاعدة البيانات في المرة الأولى للمحاولة فقط.

```
      Dim frmStatusMessage As New frmStatus()
      If Not DidPreviouslyConnect Then
         frmStatusMessage.Show("الاتصال مع خادم SQL ")
      End If
```

٥. محاولة الاتصال مع SQL Server على الكمبيوتر المحلى.

```
      Dim IsConnecting As Boolean = True
      While IsConnecting
         Try
```

٦. تكوين كائن اتصال مع SQL Server.

```
            Dim northwindConnection As New SqlConnection(strConn)
```

٧. تكوين عبارة SQL لحذف الإجراء المخزن GetCategories إذا كان موجودا.

```
            Dim strSQL As String = _
               "IF EXISTS (" & _
               "SELECT * " & _
```

```
"FROM northwind.dbo.sysobjects " & _
"WHERE Name = 'GetCategories' " & _
"AND TYPE = 'p')" & vbCrLf & _
"DROP PROCEDURE GetCategories"
```

٨. تكوين أمر SqlCommand لاستخدامة فى تعليمات SQL.

```
Dim scmd As New SqlCommand(strSQL, northwindConnection)

Try
```

٩. فتح الاتصال وتركة مفتوحا حتى الانتهاء من تنفيذ كل تعليمات SQL.

```
northwindConnection.Open()
```

١٠. الاتصال ناجح ولذلك يجب الخروج من حلقة While وإغلاق نموذج Status.

```
IsConnecting = False
DidPreviouslyConnect = True
frmStatusMessage.Close()
```

١١. تنفيذ عبارة حذف الإجراء المخزن عند وجودة باستخدام وسيلة.

```
scmd.ExecuteNonQuery()
Catch expSql As SqlException
MessageBox.Show(expSql.ToString, Me.Text, _
   MessageBoxButtons.OK, MessageBoxIcon.Error)
Exit Sub
End Try
```

١٢. تكوين عبارة SQL الخاصة بتكوين إجراء GetCategories ثم تنفيذها.

```
scmd.CommandText = _
   "CREATE PROCEDURE GetCategories " & vbCrLf & _
   "AS " & vbCrLf & _
   "SELECT CategoryID, CategoryName " & _
   "FROM Northwind.dbo.Categories"

Try
   scmd.ExecuteNonQuery()

Catch expSql As SqlException
   MessageBox.Show(expSql.ToString, Me.Text, _
      MessageBoxButtons.OK, MessageBoxIcon.Error)
   Exit Sub
End Try
```

١٣. تكوين عبارة حذف إجراء GetProducts واستخدامها فى حذف الإجر موجودا.

```
scmd.CommandText = _
    "IF EXISTS (" & _
    "SELECT * " & _
    "FROM northwind.dbo.sysobjects " & _
    "WHERE Name = 'GetProducts' " & _
    "AND TYPE = 'p')" & vbCrLf & _
    "DROP PROCEDURE GetProducts"

Try
    scmd.ExecuteNonQuery()

Catch expSql As SqlException
    MessageBox.Show(expSql.ToString, Me.Text, _
        MessageBoxButtons.OK, MessageBoxIcon.Error)
    Exit Sub
End Try
```

١٤. تكوين عبارة SQL الخاصة بتكوين إجراء GetProducts وتنفيذها.

```
scmd.CommandText = _
    "CREATE PROCEDURE GetProducts " & vbCrLf & _
    "@CategoryID Int " & vbCrLf & _
    "AS " & vbCrLf & _
    "SELECT ProductID, ProductName, UnitPrice, UnitsInStock " & _
    "FROM Northwind.dbo.Products " & _
    "WHERE CategoryID = @CategoryID"

Try
    scmd.ExecuteNonQuery()

Catch expSql As SqlException
    MessageBox.Show(expSql.ToString, Me.Text, _
        MessageBoxButtons.OK, MessageBoxIcon.Error)
    Exit Sub
Finally
```

١٥. إقفال الاتصال بعد الانتهاء من تكوين الإجراءات المخزنة.

```
            northwindConnection.Close()
         End Try
      Catch exp As Exception
```

١٦. اصطياد خطأ عدم إمكانية الاتصال مع SQL Server.

```
         frmStatusMessage.Close()
         MessageBox.Show(" SQL Server لتشغيل هذا البرنامج لابد أن يكون لديك" & _
            "with the Northwind database installed.", _
            Me.Text, MessageBoxButtons.OK, MessageBoxIcon.Error)
         End
      End Try
   End While

   frmStatusMessage.Close()

   MessageBox.Show("Northwind database تم تكوين الإجراءات المخزنة بنجاح فى", _
       Me.Text, MessageBoxButtons.OK, _
      MessageBoxIcon.Information)
   HasCreatedSprocs = True
End Sub
```

١٧. إجراء الحصول على المنتجات الخاصة بإحدى المجموعات وعرضها فى شبكة بيانات.

```
Private Sub btnGetProducts_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnGetProducts.Click

   strConn = SQL_CONNECTION_STRING
```

١٨. تكوين كائن اتصال ، كائن أمر بيانات، كائن موفق بيانات، وكائن فئة بيانات.

```
   Dim scnnNorthwind As New SqlConnection(strConn)
   Dim scmd As New SqlCommand("GetProducts", scnnNorthwind)
   Dim sda As New SqlDataAdapter(scmd)
   Dim dsProducts As New DataSet()
```

١٩. إضافة المعامل المطلوب بواسطة الإجراء المخزن عن طريق مربع السرد المركب وإضافته إلى مجموعة المعاملات الخاصة بأمر البيانات.

```
   Dim sparCatID As New SqlParameter()
```

```
With sparCatID
   .ParameterName = "@CategoryID"
   .SqlDbType = SqlDbType.Int
   .Value = cboCategoriesInputParam.SelectedValue
End With

With scmd
   .Parameters.Add(sparCatID)
   .CommandType = CommandType.StoredProcedure
End With

Try
```

٢٠. تعبئة فئة البيانات باستخدام وسيلة Fill.

```
   sda.Fill(dsProducts, "Products")

Catch expSQL As SqlException
   MessageBox.Show(expSQL.ToString, Me.Text, _
      MessageBoxButtons.OK, MessageBoxIcon.Error)
   Exit Sub
End Try
```

٢١. ربط شبكة البيانات مع الجدول المطلوب فى فئة البيانات.

```
grdProducts.DataSource = dsProducts.Tables(0)
```

٢٢. ضبط خصائص شبكة البيانات.

```
With grdProducts
   .BackColor = Color.GhostWhite
   .BackgroundColor = Color.Lavender
   .BorderStyle = BorderStyle.None
   .CaptionBackColor = Color.RoyalBlue
   .CaptionFont = New Font("Tahoma", 10.0!, FontStyle.Bold)
   .CaptionForeColor = Color.Bisque
   .CaptionText = "Northwind Products"
   .Font = New Font("Tahoma", 8.0!)
   .ParentRowsBackColor = Color.Lavender
   .ParentRowsForeColor = Color.MidnightBlue
```

٢٣. حذف جميع كائنات أنماط الجداول السابق إضافتها لكى لا يحدث خطأ.

```
   .TableStyles.Clear()
End With
```

٢٤. تكوين نمط جدول وضبط الخصائص.

```
Dim grdTableStyle1 As New DataGridTableStyle()
With grdTableStyle1
    .AlternatingBackColor = Color.GhostWhite
    .BackColor = Color.GhostWhite
    .ForeColor = Color.MidnightBlue
    .GridLineColor = Color.RoyalBlue
    .HeaderBackColor = Color.MidnightBlue
    .HeaderFont = New Font("Tahoma", 8.0!, FontStyle.Bold)
    .HeaderForeColor = Color.Lavender
    .SelectionBackColor = Color.Teal
    .SelectionForeColor = Color.PaleGreen
```

٢٥. ضبط خاصية MappingName لربط النمط مع أحد الجداول في فئة البيانات.

```
    .MappingName = dsProducts.Tables(0).TableName
    .PreferredColumnWidth = 125
    .PreferredRowHeight = 15
End With
```

٢٦. تشكيل الأعمدة التي سوف تظهر في شبكة البيانات.

```
Dim grdColStyle1 As New DataGridTextBoxColumn()
With grdColStyle1
    .HeaderText = "الرمز"
    .MappingName = "ProductID"
    .Width = 50
End With

Dim grdColStyle2 As New DataGridTextBoxColumn()
With grdColStyle2
    .HeaderText = "الإسم"
    .MappingName = "ProductName"
End With

Dim grdColStyle3 As New DataGridTextBoxColumn()
With grdColStyle3
    .HeaderText = ""السعر
    .MappingName = "UnitPrice"
    .Format = "c"
```

```
    .Width = 75
    .ReadOnly = True
  End With

  Dim grdColStyle4 As New DataGridTextBoxColumn()
  With grdColStyle4
    .HeaderText = "المخزون"
    .MappingName = "UnitsInStock"
    .Width = 75
    .Alignment = HorizontalAlignment.Center
  End With
```

٢٧. إضافة أنماط الأعمدة إلى مجموعة أنماط الأعمدة فى نمط الجدول

```
  grdTableStyle1.GridColumnStyles.AddRange _
    (New DataGridColumnStyle() _
    {grdColStyle1, grdColStyle2, grdColStyle3, grdColStyle4})
  grdProducts.TableStyles.Add(grdTableStyle1)

End Sub
```

٢٨. إجراء معالجة حدث النقر على زر الحصول على أغلى عشرة منتجات.

```
Private Sub btnGet10MostExpProds_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles btnGet10MostExpProds.Click
  Dim scnnNorthwind As New SqlConnection(SQL_CONNECTION_STRING)
```

٢٩. تكوين أمر بيانات SQL وتحديد نوعة.

```
  Dim scmd As New SqlCommand("[Ten Most Expensive Products]",
  scnnNorthwind)
  scmd.CommandType = CommandType.StoredProcedure
```

٣٠. فتح الاتصال مع قاعدة البيانات وتنفيذ الأمر لتكوين كائن قراءة بيانات.

```
  scnnNorthwind.Open()
  Dim sdr As SqlDataReader =
  scmd.ExecuteReader(CommandBehavior.CloseConnection)
```

٣١. تكوين كائن StringBuilder لربط النصوص المعروضة فى مربع النص.

```
  Dim sb As New StringBuilder()
  sb.Append("اسم المنتج")
  sb.Append(vbTab)
  sb.Append(vbTab)
```

```
sb.Append(vbTab)
sb.Append(vbTab)
sb.Append("السعر")
sb.Append(vbCrLf)
sb.Append("=========================")
sb.Append(vbTab)
sb.Append("==========")
sb.Append(vbCrLf)
```

٣٢. قراءة البيانات في حلقة باستخدام قارئ البيانات.

```
While sdr.Read
    sb.Append(sdr.GetString(0))
    sb.Append(vbTab)
    sb.Append(vbTab)
    If sdr.GetString(0).Length < 20 Then
        sb.Append(vbTab)
    End If
```

٣٣. تشكيل السلسلة في صورة عملة.

```
    sb.Append(sdr.GetSqlMoney(1).ToDouble.ToString("c"))
    sb.Append(vbCrLf)
End While
```

٣٤. عرض النتائج في مربع النص.

```
    txtTenMostExpProds.Text = sb.ToString
End Sub
```

٣٥. إجراء معالجة تغيير صفحة متحكم TabControl. ويهدف الإجراء إلى التحقق من عدم استطاعة المستخدم تشغيل إجراءات التطبيق قبل تكوين الإجراءات المخزنة.

```
Private Sub tabApp_SelectedIndexChanged(ByVal sender As Object, ByVal e As
System.EventArgs) Handles tabApp.SelectedIndexChanged
    If Not HasCreatedSprocs AndAlso tabApp.SelectedTab.TabIndex > 0 Then
        MessageBox.Show("يجب تكوين الإجراءات المخزنة أولا", _
            Me.Text, MessageBoxButtons.OK, MessageBoxIcon.Stop)
        tabApp.SelectedIndex = 0
    ElseIf HasCreatedSprocs AndAlso cboCategoriesInputParam.Items.Count = 0
    Then
```

٣٦. تعبئة مربع السرد المركب (ComboBox) بالبيانات.

```
        strConn = SQL_CONNECTION_STRING

        Dim scnnNorthwind As New SqlConnection(strConn)
        Dim dsCategories As DataSet
        Dim scmd As New SqlCommand("GetCategories", scnnNorthwind)
        Dim sda As New SqlDataAdapter(scmd)

        dsCategories = New DataSet()
        scmd.CommandType = CommandType.StoredProcedure

        Try
```

٣٧. تعبئة فئة البيانات.

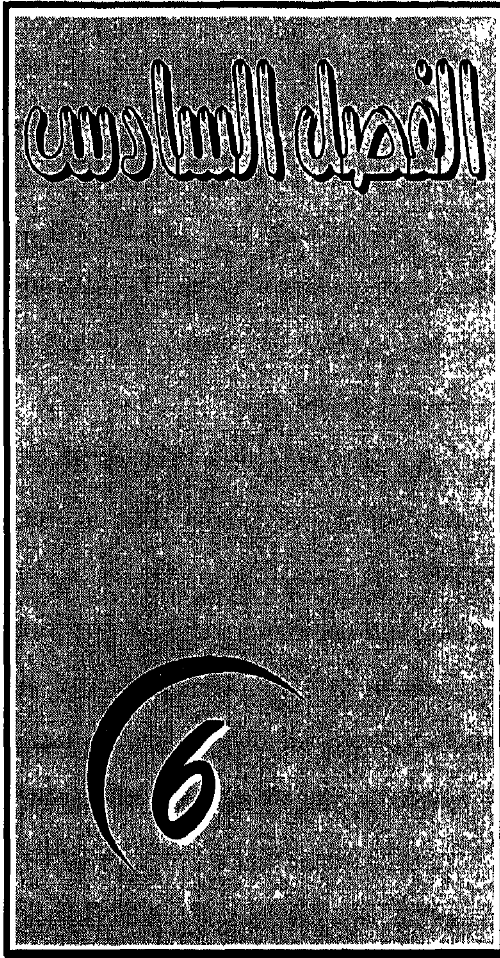
```
            sda.Fill(dsCategories)

        Catch expSql As SqlException
            MessageBox.Show(expSql.ToString, Me.Text, _
                MessageBoxButtons.OK, MessageBoxIcon.Error)
        End Try
```

٣٨. ربط البيانات مع مربع السرد المركب.

```
        With cboCategoriesInputParam
            .DataSource = dsCategories.Tables(0)
            .DisplayMember = "CategoryName"
            .ValueMember = "CategoryID"
        End With
    End If
End Sub

End Class
```



التقارير

تعتبر Crystal Reports هى الأداة الرئيسية للتقارير فى Visual Studio .NET. وتمكننا هذه الأداة من تكوين تقارير معقدة تعرض البيانات فى أشكال لا يستطيع تنفيذها غير المحترفين. ويقدم Crystal Reports للمبرمجين وسائل متكاملة داخل بيئة Visual Studio .NET. تقوم هذه الأدوات بتوفير إمكانيات رؤية وتحليل البيانات، مشاركة التقارير والمعلومات على شبكة الوب. ويتميز عرض التقارير سواء باستخدام الويندوز أو باستخدام HTML، بالتفاعل القوى مع المستخدمين عن طريق توفير وسائل، مثل التنقيب عن البيانات باستخدام الرسوم البيانية، التجول فى التقارير، والبحث عن النصوص.

وبدلا من استخدام الكود فى تكوين التقارير، يقوم مولد التقارير فى Crystal Reports بتقديم واجهات لتصميم وصياغة التقارير التى نحتاج إليها بسرعة وسهولة. يطلق على هذه الواجهات مصطلح خبْراء التقارير (Report Experts). وهى تمثل برامج يمكن عن طريقها الاختيار من بين مجموعة متنوعة من التقارير التى تبدأ من التقارير القياسية إلى إعداد الخطابات، عرض الرسوم البيانية التى تمكن المستخدم من تتبع البيانات والوصول إلى التفصيلات، حساب الإجماليات و الإجماليات الفرعية والنسب المئوية. ولقد تم تصميم مولد تقارير Crystal Reports لكى يصبح جزءا متكاملا مع باقى الأجزاء فى بيئة Visual Studio.NET.

وتتميز تقارير Crystal Reports بإمكانية استخدامها فى بيئة الوب وفى بيئة الويندوز. كما يمكن إصدار تقارير Crystal Reports فى صورة خدمات تقارير من على خادم وب. على سبيل المثال، يمكن تكوين تطبيق وب يتيح للمستخدم التنقيب فى المعلومات الخاصة برسم بياني وترشيح هذه البيانات بما يتوافق مع احتياجاته.

أدوات تكوين التقارير

يحتوى Crystal Reports على أنواع مختلفة من الأدوات المطلوبة لتكوين التقارير. يشمل ذلك أدوات تصميم التقارير، أدوات مشاهدة التقارير، وأدوات الوصول إلى مصادر البيانات.

أدوات تصميم التقارير

تستخدم أدوات تصميم التقارير فى تخطيط أقسام التقرير، تحديد البيانات التى يتم

وضعها على صفحات التقرير، وأدوات صياغة التقرير وتحديد شكل عرض التقرير. تشمل هذه الأدوات مصمم التقارير، الذى يمثل الأداة الرئيسية المستخدمة فى صناعة التقارير. كما يشمل واجهات المساعدة فى تصميم التقرير، والتى يطلق عليها خبراء التقارير (Report Experts).

مصمم تقارير Crystal Reports

ينقسم مصمم تقارير Crystal Report Designer إلى الأقسام التالية :

- مصمم التقرير (Report Designer).
- شريط الأدوات (Crystal Reports Toolbars).
- مستكشف الحقول (Field Explorer).

مصمم التقرير

ينقسم التقرير فى نافذة مصمم التقرير، إلى خمسة أقسام رئيسية. ويمكن إخفاء بعض هذا الأقسام أو إضافة أقسام أخرى. تشمل هذه الأقسام: مقدمة التقرير، مقدمة الصفحة، قسم التفصيلات، مؤخرة التقرير، مؤخرة الصفحة. وتتوقف البيانات التى تظهر بالتقرير النهائي على خيارات التصميم؛ وتتوقف خصوصا على أنواع الأقسام التى نختارها لإدراج كائنات تقرير معينة. على سبيل المثال، إذا أدرجنا كائن رسم بياني فى قسم مقدمة التقرير، سوف يظهر الرسم مرة واحدة فى بداية التقرير وسوف يلخص البيانات التى يحتوى عليها التقرير. بدلا من ذلك، إذا تم إضافة كائن تقرير إلى قسم مقدمة مجموعة، سوف يظهر فى بداية كل مجموعة رسم بياني منفصل وسوف يلخص البيانات المتعلقة فقط بتلك المجموعة.

مقدمة التقرير

يجرى طباعة الكائنات الموضوعة فى قسم مقدمة التقرير مرة واحدة فى بداية التقرير. ويحتوى قسم مقدمة التقرير بصفة عامة على عنوان التقرير والمعلومات الأخرى التى نريد إظهارها فقط فى بداية التقرير. وتحتوى الرسوم البيانية والجداول المتقاطعة التى توضع فى هذا القسم على بيانات تخص كامل التقرير. كما أن الصيغ الموضوعة فى هذا القسم يتم تقييمها مرة واحدة فى بداية التقرير.

مقدمة الصفحة

الكائنات التى توضع فى مقدمة الصفحة يتم طباعتها فى بداية كل صفحة. ويحتوى قسم مقدمة الصفحة بصفة عامة على المعلومات التى نريد طباعتها فى قمة كل صفحة. يمكن أن يشمل ذلك حقول النصوص مثل أسماء الفصول، اسم الوثيقة، أو المعلومات المشابهة الأخرى. ويمكن استخدام هذا القسم أيضا ليحتوى على عناوين الحقول، التى سوف تعرض فى هذه الحالة فى صورة تعريفات لأعمدة حقول البيانات فى التقرير. لايمكن وضع الرسوم البيانية والجداول المتقاطعة فى هذا القسم. كما يجرى تقييم الصيغ التى توضع فى هذا القسم مرة واحدة فى بداية كل صفحة جديدة.

قسم التفصيلات

يحتوى هذا القسم على البيانات التى تكون جسم التقرير، وهو القسم الذى تظهر به معظم بيانات التقرير. ويجرى طباعة الكائنات التى توضع فى هذا القسم بالنسبة لكل سجل من سجلات البيانات. على سبيل المثال، إذا أضفنا كائن قاعدة بيانات إلى هذا القسم، ويحتوى هذا الكائن على ١٠٠ سجل، سوف يقوم التقرير بطباعة مائة قسم تفصيلات فى وقت التشغيل. ولا يمكن وضع الرسوم البيانية والجداول المتقاطعة فى هذا القسم. كما أن الصيغ الموضوعة فى هذا القسم يتم تقييمها مرة بالنسبة لكل سجل.

مؤخرة التقرير

الكائنات الموضوعة فى مؤخرة التقرير، تطبع مرة واحدة فى نهاية ذلك التقرير. ويستخدم هذا القسم لكى يحتوى على المعلومات التى نريد إظهارها مرة واحدة فى نهاية التقرير، مثل الإجماليات النهائية. وتحتوى الرسوم البيانية والجداول المتقاطعة الموضوعة فى هذا القسم على بيانات لكل التقرير. كما أن الصيغ الموضوعة فى هذا القسم، يتم تقييمها مرة واحدة فى نهاية التقرير.

مؤخرة الصفحة

الكائنات الموضوعة فى قسم مؤخرة الصفحة ، يتم طباعتها فى نهاية كل صفحة. وفى العادة ، يحتوى هذا القسم على رقم الصفحة وأي معلومات أخرى نريد طباعتها فى نهاية كل صفحة. ولا يمكن وضع الرسوم البيانية والجداول المتقاطعة فى هذا القسم. كما يتم تقييم

الصيغ الموضوعة فى هذا القسم فى نهاية كل صفحة جديدة.

مقدمة المجموعة

يظهر هذا القسم عند إضافة مجموعة إلى التقرير، قبل قسم التفصيلات مباشرة. ويتم طباعة الكائنات الموضوعة فى هذا القسم فى بداية كل مجموعة جديدة. ويحتفظ هذا القسم فى الأساس بحقل اسم المجموعة، كما يمكن أيضا إستخدامة لعرض الرسوم البيانية التى تشتمل على بيانات خاصة بالمجموعة. ويتم تقييم الصيغ الموضوعة فى هذا القسم فى بداية المجموعة.

قسم ذيل المجموعة

يظهر هذا القسم عند إضافة مجموعة إلى التقرير، بعد قسم التفصيلات مباشرة. ويتم طباعة الكائنات التى يحتوى عليها فى نهاية كل مجموعة. ويحتوى هذا القسم بصفة عامة على قيم الإجماليات الفرعية، ويمكن أن يستخدم فى عرض الرسوم البيانية أو الجداول المتقاطعة. ويتم تقييم الصيغ الموضوعة فى هذا القسم فى نهاية كل مجموعة.

وتتوقف البيانات التى تظهر فى التقرير التام على خيارات التصميم التى يحددها المبرمج. وعلى وجه الخصوص، تتنوع بيانات التقرير على أساس الأقسام التى نختارها لإدراج كائنات التقرير المختلفة. على سبيل المثال ، إذا أدرجنا كائن رسم بياني فى قسم مقدمة التقرير (Header)، فإن الرسم سوف يظهر لمرة واحدة فقط فى بداية التقرير شاملا البيانات التى يحتوى عليها التقرير. وبدلا من ذلك ، إذا تم إضافة كائن رسم بياني إلى قسم مقدمة مجموعة ، فإن رسم بياني منفصل سوف يظهر فى بداية كل مجموعة بيانات شاملا البيانات المتعلقة بالمجموعة فقط.

نافذة *Field Explorer*

يستخدم مربع Field Explorer لإدراج، تعديل، أو حذف حقول تقرير Crystal. يعرض هذا المربع شجرة تتكون من حقول قاعدة بيانات وحقول خاصة يمكن إضافتها إلى التقرير. ويبين مربع Field Explorer أيضا الصيغة، المعامل، اسم المجموعة، الإجمالي المتحرك، والحقول غير المرتبطة بأدوات تحكم التى سبق تعريفها للاستخدام فى التقرير. ويعرض مربع Field Explorer علامات اختيار بجانب الحقول التى تم إضافتها إلى التقرير، وبجانب

الحقول المستخدمة بواسطة حقول أخرى أو المستخدمة فى العمليات الحسابية، مثل حقول الصيغ، المجموعات، الإجماليات المتحركة، والملخصات. ولمشاهدة هذا المربع:

١. نشير إلى Other Windows فى قائمة View ثم نختار Document Outline.

٢. ننقر بزر الماوس الأيمن على أى حقل لمشاهدة القائمة المختصرة الخاصة باختياراته.

أشرطة أدوات *Crystal Reports*

عند فتح مصمم Crystal Reports لتكوين أو تعديل أحد التقارير، يظهر تلقائيا اثنان من شرائط الأدوات:

١. شريط Main Toolbar الذى يحتوى على أزرار خاصية باختيار مربعات حوار، مثل Toggle Field View، Select Expert، TopN Expert، Sort Order، Object Properties. وتوجد بالشريط خيارات الصياغة الأساسية لكائنات النصوص وقائمة Zoom منسدلة.

٢. شريط Insert Toolbar الذى يحتوى على خيارات الإدراج المختلفة، مثل Insert Group، Insert Summary، Insert Subreport، Insert Chart، و Insert Picture

لإظهار أو إخفاء شرائط أدوات Crystal Reports نشير إلى Toolbars فى قائمة View، ثم ننقر على اسم القائمة المطلوبة.

خبراء التقارير

يمكن استخدام مصمم Crystal Reports لتكوين تقرير بدون استخدام أدوات صناعة التقارير المتوفرة بالنظام، كما يمكن استخدام أدوات خبراء التقارير (Crystal Reports Experts) للمساعدة فى عمليات التصميم. فيما يلى قائمة بخبراء التقارير التى يحتوى عليها Crystal Reports:

١. تقرير قياسي (Standard).

٢. نموذج الخطاب (Form Letter).

٣. نموذج (Form).

٤. جداول متقاطعة (Cross-tab).

٥. تقرير فرعي (Subreport).

٦. عنوان بريدي (Mail Label).

٧. تتبع التفصيلات (Drill Down).

كل أداة من هذه الأدوات تقودنا خلال عملية تكوين التقرير عن طريق تزويدنا بسلسلة من الجداول. ويحتوى الكثير من هذه الأدوات على جداول مساعدة خاصة بأنواع معينة من التقارير.

خبير التقرير القياسى (*Standard Report Expert*)

تعتبر هذه الأداة هى الأكثر استخداما بين جميع أدوات Crystal Reports Experts. وتحتوى على ثمانية جداول، الكثير منها شائع الاستخدام بواسطة أدوات Experts الأخرى. ويقوم خبير التقرير القياسى بقيادتنا خلال عملية اختيار مصدر البيانات والربط مع جداول البيانات. كما يساعدنا فى إضافة الحقول وتحديد المجموعات، تكوين الإجماليات، تحديد معيار الفرز الذى نريد إستخدامة. وفى النهاية، يقودنا Standard Report Expert نحو تكوين الرسوم البيانية واختيار السجلات. وترتبط هذه الأداة بمربع حوار به ملصقات مختلفة ترتبط بصفحات لتحديد الخيارات التى سوف يتم استخدامها. من بين هذه الملصقات، ملصق Style الذى يحتوى على مخططات سابقة الإعداد يمكن تطبيقها على التقرير لجعلة أكثر تأثيرا على المشاهد.

خبير إعداد الخطابات (*Form Letter Report Expert*)

يقدم هذا الخبير حلا بسيطا لتكوين الخطابات التى تستخدم قاعدة البيانات باعتبارها مصدرا للمعلومات عن العميل. ومع أن هذا الخبير يقدم الكثير من الوظائف التى يقدمها خبير التقرير القياسى، إلا أنة يقدم جدولا خاصا بتقارير الخطابات يساعد فى تعريف النص وحقول قاعدة البيانات التى تظهر فى كل قسم من أقسام الخطاب. ويمكن أيضا استخدام جدول الخطابات لاستيراد نص سبق تكوينه فى تطبيق أخر.

خبير إعداد النماذج (Form Report Expert)

يتيح لنا خبير النموذج تكوين تقارير يمكن طباعتها على نماذج سابقة التجهيز (فواتير الشركة، كشوف الحسابات، وما يماثلها). ومع أنة يماثل خبير التقارير العادية إلى حد كبير، إلا أنة يحتوى على جدول خاص بالنماذج السابق تجهيزها تتيح لنا اختيار أشكال تتناسب مع شعار الشركة.

خبير الجداول المتقاطعة (Cross-tab Report Expert)

تقودنا هذه الأداة خلال عملية تكوين تقرير يتم فيه عرض البيانات فى صورة متقاطعة. ومن بين الملصقات التى يحتوى عليها مربع حوار Cross Tab Report Expert ملصق Style، ملصق Cross-tab، و ملصقCustomize Style، التى تساعدنا على تكوين وصياغة البيانات المتقاطعة ذاتها.

خبير التقارير الفرعية (SubReport Expert)

يتيح لنا خبير التقارير الفرعية تكوين تقرير رئيسي وتقرير فرعى فى نفس الوقت. وتقدم هذه الأداة المرونة المتوفرة فى خبير التقارير القياسى لتكوين التقرير الرئيسى. وفى جدول التقرير الفرعى الإضافي، يمكننا اختيار استخدام أحد التقارير الموجودة على أنة تقرير فرعى، أو يمكننا تكوين تقرير جديد. ولكى نكون تقرير فرعى نتبع نفس الخطوات المتبعة فى خبير التقارير القياسى.

خبير ملصقات عناوين البريد (Mail Label Report Expert)

يتيح لنا هذا الخبير تكوين تقرير خاص بالطباعة على ملصقات عناوين البريد من أى حجم. وللقيام بذلك، نستخدم صفحة Label فى مربع حوار Mailing Label Expert، لاختيار نوع الملصق التجاري، كما يمكننا تعريف مخططاتنا من الصفوف والأعمدة لأى نمط من التقارير متعددة الأعمدة.

خبير تقرير التنقيب (Drill Down Report Expert)

تسمح لنا هذه الأداة بتكوين تقرير يقوم بإخفاء بعض الأقسام ويجعلها متاحة للمشاهدة فقط من خلال عملية التنقيب (Drill Down). ويقدم لنا هذا الخبير كل وظائف خبير

التقارير القياسية. ومن أهم الملصقات التى يحتوى عليها مربع حوار Drill Down Report Expert، ملصق Drill الذى يعرض قائمة بالأقسام التى يمكن إخفاؤها. ولا تظهر الأقسام المختفية إلى أن يتم النقر على الحقل المناسب. ويمكن استخدام البيانات المخفية فى الملخصات والمجاميع.

أدوات الوصول إلى التقارير ومصادر البيانات

يحتاج المستخدم الوصول إلى التقرير لقراءته، كما يحتاج التقرير الوصول إلى مصدر البيانات للحصول على البيانات. الأداة الرئيسية التى توفر للمستخدم الوصول إلى التقرير هى أداة مشاهدة التقارير بنماذج الويندوز (Forms Viewer). كما يمكن للمستخدم الوصول إلى التقارير عن طريق تصديرها إلى صيغة ملائمة للمستخدم أو عن طريق طباعة هذه التقارير. من ناحية أخرى، يستطيع التقرير الوصول إلى مصادر البيانات باستخدام محركات قواعد البيانات (Database Drivers) التى يحتوى عليها Crystal Reports.

أداة مشاهدة التقارير

تستخدم أداة مشاهدة تقارير نماذج الويندوز لعرض تقارير Crystal Reports أمام المستخدمين على شاشة الكمبيوتر. ويجب ربط هذه الأداة مع التقارير لكى يمكن استخدامها. ويمكن برمجة التفاعل بين أداة Windows Forms Viewer وبين أدوات التحكم الأخرى فى التطبيق. بالإضافة إلى ذلك، يأتى مع هذه الأداة شريط أدوات خاص بها، يمكن تعديله بما يتفق مع الاحتياجات المختلفة للتطبيق. وتوجد أداة Viewer ضمن أدوات التحكم الموجودة فى مربع ToolBox تحت اسم CrystalReportViewer. ويحتوى هذا المتحكم على مجموعة من الخصائص التى تسمح لنا بالتحكم فى شكل وسلوك التقرير.

تصدير التقارير

يقصد بتصدير التقرير تحويلة إلى صيغة من الصيغ الأخرى للتقارير. ويتيح لنا Crystal Reports تصدير التقارير إلى الصيغ التالية :

- Adobe Acrobat (.pdf)
- Crystal Reports for Visual Studio .NET(.rpt)
- HTML 3.2 an 4.0 (.html)
- Microsoft Excel(.xls)

- Microsoft Rich Text(.rtf)
- Microsoft Word(.doc)

ويدعم متحكم Crystal Report Viewer تصدير التقارير من خلال استخدام زر Export بشريط الأدوات الخاص به، إلى كل صيغ التصدير المذكورة فيما عدا HTML و Crystal Reports for Visual Studio.NET. ويمكن استخدام الكود لضبط خيارات تصدير التقارير.

طباعة التقارير

يمكن تزويد المستخدمين بخيارات الطباعة من خلال شريط أدوات متحكم Report Viewer. كما يمكن إضافة خيار الطباعة إلى متحكم يتفاعل مع مشاهد التقرير. ويمكن أيضا تكوين تطبيق يقوم أتوماتيكيا بطباعة تقارير Crystal Reports على طابعة محددة أو على طابعة افتراضية.

محركات البيانات

تستخدم محركات البيانات فى وقت التصميم للتعرف على مخططات البيانات. ولكى يتم تحديد هذه المخططات، يجب إجراء اتصال مع قواعد البيانات من خلال استخدام محرك OLEDB، محرك ODBC، ومحركات EXCEL/ACCESS للحصول على الجداول، المشاهد، والإجراءات المخزنة. كما يتم الاتصال مع محرك ADO.NET للحصول على مخطط البيانات فى صورة ملف XML. وفى وقت التشغيل، تستخدم التقارير نفس محركات البيانات المستخدمة فى إعداد التقارير. وإذا تم دفع البيانات إلى Crystal Reports، سوف يجرى استخدام المحرك المناسب لمعالجة فئة السجلات أو فئة البيانات. ويأتي Crystal Reports ومعه عدد من محركات البيانات التى تعمل فى نظام Visual Studio.NET.

الجدول رقم (٢٦) يعرض محركات قواعد البيانات المشحونة مع Crystal Reports.

محرك قاعدة البيانات	قاعدة البيانات المستخدم معها
OLEDB	أى قاعدة بيانات تدعم OLEDB
ODBC	أى قاعد بيانات بها محرك ODBC
Access/Excel	مايكروسوفت Access ومايكروسوفت Excel

محرك قاعدة البيانات	قاعدة البيانات المستخدم معها
ADO.NET	قواعد البيانات التى تدعم هذه التقنية
Field Definition	لا يستخدم مع قواعد البيانات
CDO	لا يستخدم مع قواعد بيانات

جدول ٢٦

تصميم تطبيقات التقارير

يتناول هذا القسم موضوعات تصميم التطبيقات التى تحتوى على تقارير. تشمل هذه الموضوعات عمليات تصميم التقرير التى تتمثل فى تحديد أقسام التقرير المستخدمة والكائنات التى تحتوى عليها وما يتعلق بذلك من ربط التقارير مع مصادر البيانات ومع الأدوات التى تمكن من استخدامها. كما تشمل موضوعات تصميم تطبيقات التقارير أيضا، عرض كيفية التعامل مع كائنات التقرير المتمثلة فى الحقول والنصوص المختلفة الموضوعة فى أقسام التقرير.

تصميم التقرير

يمكن تصميم التقارير عن طريق الاستعانة بأدوات خبراء التقارير (Report Experts) التى يوفرها Crystal Reports، كما يمكن تصميم التقارير بدون استخدام هذه الأدوات. ويعتبر مصمم التقارير (Report Designer) هو ورشة العمل التى يتم خلالها تصميم التقارير المختلفة. وتشمل عملية التصميم بدء تقرير جديد، تحديد مصادر البيانات التى يستخدمها التقرير، ربط التقرير مع مصادر البيانات، وضع الكائنات فى أقسام التقرير المختلفة، وربط التقرير مع أداة مشاهدة التقارير بنماذج الويندوز.

بدء التقرير وتحديد مصادر البيانات

أول مهام تصميم التقرير هى تحديد مصادر البيانات المستخدمة لتوريد البيانات التى يحتوى عليها التقرير. وهناك العديد من مصادر البيانات التى يمكن استخدامها مع تقارير

Crystal Reports. غير أن أهم ما تتميز به التقارير في Visual Basic .NET هو استخدام فئات البيانات (Datasets) مصدرا لهذة التقارير.

بدء تكوين تقرير جديد

١. فـى نـافذة Solution Explorer، نـنقر بـزر الماوس الأيمن على المشروع لعرض قائمة مختصرة.

٢. نشير إلى Add وننقر على Add New Item.

٣. فـى مربع حوار Add New Item، نختار Crystal Report من منطقة Templates ثم نـنقر Open. يترتـب عـلى ذلـك عـرض مربع حوار Crystal Report Gallery المبين بالشكل رقم (١٩).

٤. في مربع حوار Crystal Report Gallery، نختار واحدا من الخيارات التالية :

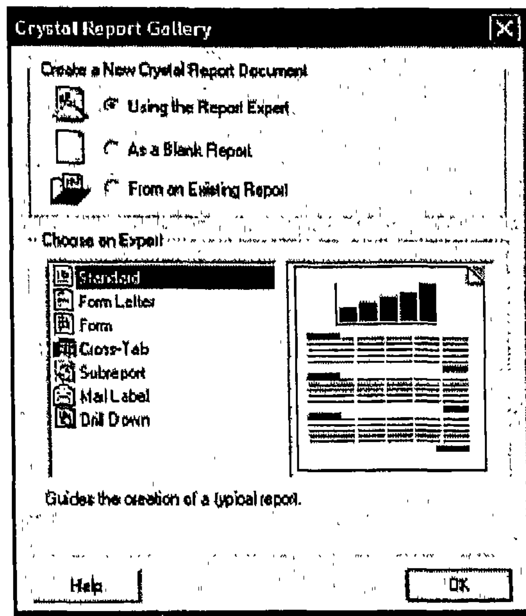
أ. اسـتخدام خـبير الـتقارير (Report Expert) – يقودنـا خلال عملية تكوين التقرير وإضافة اختياراتنا إلى مصمم تقرير Crystal.

ب. استخدام تقرير خالي – يفتح لنا مصمم Crystal Reports.

ت. اسـتخدام تقرير موجود – يؤدى الى تكوين تقرير جديد بنفس تصميم تقرير أخر.

٥. ننقر OK.

عند اختيار خبير التقارير، يظهر مربع حوار Report Expert. نختار البيانات المطلوبة لكل مجلد، نعمل خلال واجهة جدولة Report Expert، وننقر Finish للوصول إلى مصمم Crystal Report والتقرير الخاص بنا.



شكل رقم ١٩

مصادر البيانات التي يمكن استخدامها

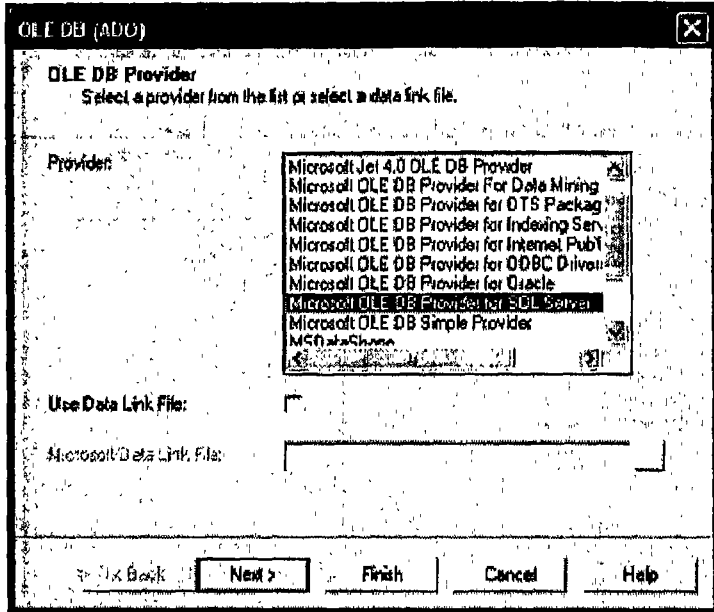
يقوم Crystal Reports بالاتصال مع قواعد البيانات من خلال محركات قواعد البيانات (database drivers). ويتم تخصيص كل محرك للتعامل مع نوع معين من قواعد البيانات أو تكنولوجيا وصول معينة إلى قاعدة البيانات. ويمكن أن يقوم Crystal Reports بالوصول إلى وإعداد التقارير من مصادر البيانات التالية:

- أي قاعدة بيانات تستخدم محرك ODBC
- أي قاعدة بيانات تستخدم محرك OLEDB
- قواعد بيانات Microsoft Access
- كتب عمل Microsoft Excel
- فئات بيانات ADO.NET
- فئات سجلات ADO
- فئات سجلات CDO

- فئات سجلات DAO
- فئات سجلات RDO

اختيار مصدر بيانات التقرير

يمكن ربط كائنات التقارير مع مصادر البيانات، كما يمكن تغيير مصدر البيانات الذى يستخدمه التقرير بعد إجراء الاتصال بين كائن التقرير وبين مصدر البيانات. وفى حالة استخدام التقرير لأكثر من جدول بيانات، يمكن ربط الجداول معا، كما يمكن تغيير هذه الروابط بعد ذلك. تبين الموضوعات التالية، كيفية اختيار مصدر بيانات، كيفية الربط بين الجداول التى يتكون منها مصدر بيانات التقرير، وكيفية إدراج الحقول فى التقارير.



شكل رقم ٢٠

اختيار مصادر البيانات وربطها بالتقرير

١. فى مربع Field Explorer، ننقر بزر الماوس الأيمن على بند Database Fields ثم نختار بند Add/ Remove Database.

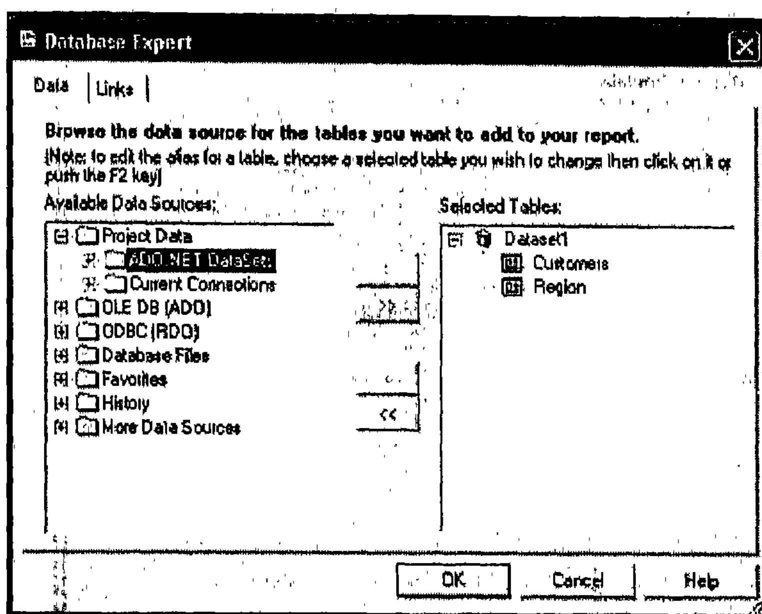
٢. فى حالة عدم وجود اتصال مع مصدر بيانات، يجب تكوين اتصال بالنقر على

أحد مصادر البيانات ثم اختيار مورد البيانات المطلوب من مربع حوار مورد البيانات. الشكل رقم (٢٠) يحتوى على موردى بيانات OLE DB.

٣. فى مربع حوار Database Expert المبين بالشكل رقم (٢١)، نتصفح المجلدات إلى أن نجد جداول مصدر البيانات المستخدم.

٤. نختار الجداول التى تحتوى على البيانات التى نريد إعداد تقرير منها ثم ننقر على الزر الخاص

٥. بإضافتها إلى جانب Selected Tables فى مربع حوار Database Expert، ثم ننقر Ok.



شكل رقم ٢١

الكود التالى يقوم بضبط معلومات الاتصال الخاصة بكل جدول من الجداول التى يحتوى عليها تقرير:

```
Imports CrystalDecisions.CrystalReports.Engine
Imports CrystalDecisions.Shared

Public Class Form1
```

```
    Inherits System.Windows.Forms.Form

    Dim crReportDocument As CrystalReport1
    Dim crDatabase As Database
    Dim crTables As Tables
    Dim crTable As Table
    Dim crTableLogOnInfo As TableLogOnInfo
    Dim crConnectionInfo As ConnectionInfo

#Region " Windows Form Designer generated code "

    Public Sub New()
        MyBase.New()
        InitializeComponent()

        crReportDocument = New CrystalReport1()
        crConnectionInfo = New ConnectionInfo()
        With crConnectionInfo
            .ServerName = "FHOME1"
            .DatabaseName = "Pubs"
            .UserID = "sa"
            .Password = "fnasr"
        End With
        crDatabase = crReportDocument.Database
        crTables = crDatabase.Tables

        For Each crTable In crTables
            crTableLogOnInfo = crTable.LogOnInfo
            crTableLogOnInfo.ConnectionInfo = crConnectionInfo
            crTable.ApplyLogOnInfo(crTableLogOnInfo)
        Next
        CrystalReportViewer1.ReportSource = crReportDocument

    End Sub
```

تعريف الجداول المفترضة بناءا على أمر/ استعلام *SQL*

إذا كانت قاعدة البيانات التي نستخدمها تدعم لغة استعلام مثل SQL، يمكن كتابة أوامر SQL الخاصة بنا عن طريق استخدام عقدة Add Command في مربع حوار Database Expert. ويترتب على ذلك، قيام Crystal Reports بتكوين كائن جدول افتراضي يمثل هذه

الأوامر. يمنح ذلك مستخدمي قاعدة البيانات المتمرسين القدرة على إحكام الرقابة على معالجة البيانات التى تدفع إلى خادم قاعدة البيانات. لتكوين كائن أمر SQL خاص بالمستخدم:

١. ننقر بزر الماوس الأيمن على Database Fields فى مربع Field Explorer ثم ننقر Add/Remove Database.

٢. فى مربع حوار Database Expert، نتصفح المجلدات إلى أن نصل إلى مصدر البيانات الخاص بنا.

٣. أسفل مصدر البيانات، ننقر نقرا مزدوجا على عقدة Add Command.

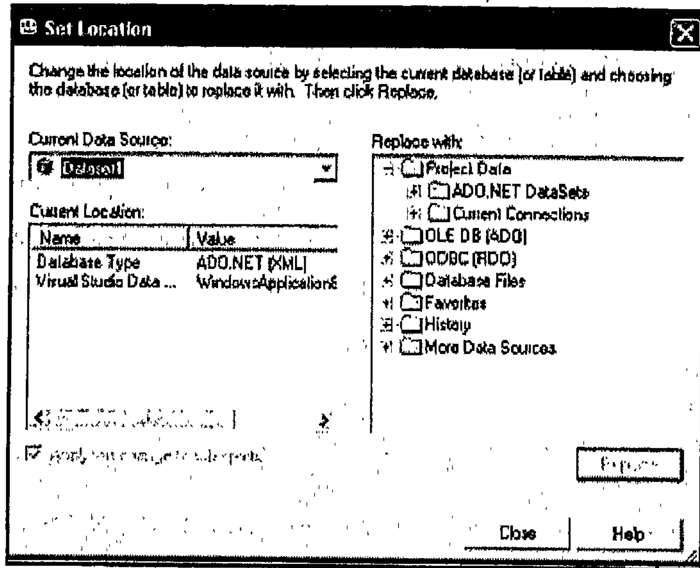
٤. فى مربع حوار Add Command to Report، ندخل الأمر أو الاستعلام المناسب لمصدر البيانات باستخدام كود مماثل للكود التالي:

```
SELECT
   Customer.`Customer ID`,
   Customer.`Customer Name`,
   Customer.`Last Year's Sales`,
   Customer.`Region`,
   Customer.`Country`,
   Orders.`Order Amount`,
   Orders.`Customer ID`,
   Orders.`Order Date`
FROM
   Customer INNER JOIN Orders ON
      Customer.`Customer ID` = Orders.`Customer ID`
WHERE
   (Customer.`Country` = 'USA' OR
   Customer.`Country` = 'Canada') AND
   Customer.`Last Year's Sales` < 10000.
ORDER BY
   Customer.`Country` ASC,
   Customer.`Region` ASC
```

٥. ننقر OK. يترتب على ذلك العودة إلى مصمم التقرير، وظهور الجدول الإفتراضى فى مربع Field Expolorer.

تغيير موقع مصدر البيانات

نستخدم أمر Set Location للإشارة إلى الاسم أو الموقع الجديد للجداول النشطة في التقرير. ويعتبر ذلك مفيدا عند استقبال تقرير يستخدم قاعدة بيانات توجد بموقع مختلف على النظام، أو عند تغيير الدليل أو موقع الاسطوانة الخاصة بقاعدة البيانات. بالإضافة إلى ذلك، يقوم أمر Set Location أتوماتيكيا بتحويل محرك البيانات الذى نستخدمه إلى مصدر البيانات الذى نقوم باختياره. على سبيل المثال، يمكننا تغيير مصدر بيانات وصول مباشر إلى مصدر بيانات ODBC باستخدام هذا الأمر.



شكل رقم ٢٢

لتغيير موقع مصدر بيانات:

١. ننقر بزر الماوس الأيمن فى مصمم التقرير، نشير إلى Database، ثم ننقر Set Location. يؤدى ذلك إلى عرض مربع حوار Set Location، المبين بالشكل رقم (٢٢).

٢. من قائمة Current Data source list، نختار جدول مصدر البيانات الذى نريد تغييره.

٣. فى قائمة Replace with ، نتصفح إلى أن نصل إلى مصدر البيانات الجديد.

٤. نختار الجدول الذى نريد التغيير إلية.

٥. ننقر Replace.

٦. ننقر Close بعد الانتهاء من تغيير كل الجداول التى نريد تغييرها.

ربط جداول البيانات

يمكن ربط الجداول التى تشكل مصدر بيانات لتقرير Crystal، باستخدام حقل مشترك بين جدولين. ويستخدم Crystal Reports هذا الرباط لموافقة السجلات من جدول مع السجلات من جدول أخر. على سبيل المثال، يمكن الربط بين جدول Orders وجدول Customers لكى يمكن تخصيص كل أمر فى جدول Orders لأحد العملاء فى جدول Customers.

وللقيام بعملية الربط المذكورة، نستخدم ملصق Link فى مربع حوار Database Expert للربط بين جداول قاعدة البيانات. وأسهل الطرق للربط بين الجداول هو اختيار Auto-Link. تقوم Auto-Link أتوماتيكيا باختيار الروابط المناسبة للجداول على أساس الحقول المشتركة فى هذه الجداول أو حقول الفهرسة (Indexed Fields). ويمكن أيضا الربط اليدوى بين جداول قاعدة البيانات. وإذا كان لدينا العديد من الروابط، يمكننا استخدام Order Links لترتيب هذه الروابط.

للربط بين الجداول التى تشكل مصدر بيانات تقرير:

١. فى مربع Field Explorer، ننقر بزر الماوس الأيمن على Database Fields ثم نختار Visual Linking Expert.

٢. فى مربع حوار Database Expert، ننقر Auto-Link للربط الأتوماتيكي بين جداول قاعدة البيانات، أو ننقر ونسحب حقل بيانات من أحد الجداول إلى أخر لتكوين رباط يدوى.

٣. ننقر Order Links لترتيب الروابط بالتسلسل الذى نريد معالجتها به.

٤. ننقر OK.

إدراج حقول قاعدة بيانات فى التقرير

يقوم Crystal Reports بعرض كل حقول قاعدة البيانات المتاحة فى مربع Field Explorer. ولإدراج أحد الحقول بالتقرير:

١. فى مربع Field Explorer، نوسع عقدة Database Fields لكى نتمكن من مشاهدة جداول قاعدة البيانات.

٢. نوسع جدول قاعدة البيانات ونختار حقل البيانات المستهدف. ويمكن مشاهدة قيمة الحقل الذى يتم اختيارة وكذلك نوع وحجم الحقل بالنقر بزر الماوس الأيمن ثم اختيار Browse Data من القائمة المختصرة. يعرض مربع الحوار الناتج فئة فرعية من قيم الحقل، مع اسم الحقل، نوعة، وطولة.

٣. نسحب الحقل الذى تم اختيارة إلى قسم Details أو أى قسم أخر بالتقرير.

استخدام فئات البيانات فى إعداد التقارير

قبل إعداد التقارير التى تستخدم فئات البيانات (Datasets)، يجب تكوين كائن فئة بيانات ثم الاتصال مع ذلك الكائن. وبالنظر إلى أن كائن فئة البيانات لا يحتوى على البيانات أثناء التصميم، لذلك لايمكن تصفح بيانات الحقول فى مصمم Crystal Reports. ولتمكين التقرير من عرض البيانات الحقيقية فى وقت التشغيل، يجب أولا دفع البيانات إلى كائن فئة البيانات (Datasets) ثم ربط فئة البيانات مع كائن تقرير..

تكوين كائن فئة بيانات

كائن فئة بيانات ADO.NET هو عبارة عن مثل من تصنيف DataSet يتم تكوينه فى الذاكرة ويعتبر صورة مصغرة لقاعدة بيانات. ويمكن تكوين فئة بيانات ADO.NET من مصادر بيانات متنوعة؛ مثل قاعدة بيانات Access، Oracle، و SQL Server عن طريق استخدام مصمم فئات البيانات فى ADO.NET.

لتكوين كائن فئة بيانات من قاعدة بيانات:

١. نكون ملف مخطط جديد فى المشروع:

أ. فى نافذة Solution Explorer، ننقر بزر الماوس الأيمن على اسم المشروع،

نشير إلى Add، ثم ننقر Add New Item.

ب. فى جانب Categories بمربع حوار Add New Item، نوسع المجلد ونختار Data.

ت. فى جانب Templates نختار Dataset.

ث. نقبل الاسم الإفتراضى Dataset1.xsd. يؤدى ذلك إلى تكوين ملف المخطط الجديد الذى سوف يستخدم لتكوين فئة البيانات النوعية. وسوف يعرض ملف المخطط فى مصمم فئات بيانات ADO.NET.

٢. نحدد موقع قاعدة البيانات:

أ. فى نافذة Server Explorer، ننقر بزر الماوس الأيمن على Data Connections ونختار Add Connection.

ب. فى مربع حوار Data Link Properties، ننقر ملصق Provider ونختار مورد بيانات. على سبيل المثال، نختار Microsoft OLE DB Provider for SQL Server.

ت. ننقر على ملصق Connection ونختار موقع قاعدة البيانات. يشمل ذلك إدخال اسم الخادم ومعلومات المرور عند الضرورة.

ث. ننقر OK. يترتب على ذلك، ظهور قاعدة البيانات، جداولها، وحقولها فى نافذة Server Explorer تحت عقدة Data Connection.

٣. فى نافذة Solution Explorer، نقوم بالنقر المزدوج على ملف Dataset1.xsd لفتحة فى مصمم XML، إذا لم يكن مفتوحا.

٤. لبناء مخطط لفئة البيانات، نسحب الجداول التى نريد استخدامها من نافذة Server Explorer إلى نافذة ملصق Dataset فى ملف Dataset1.xsd.

٥. فى قائمة File، ننقر على Save Dataset1.xsd لحفظ ملف Dataset1.xsd.

٦. فى قائمة Build، ننقر Build لتوليد كائن فئة البيانات بالمشروع.

الاتصال مع كائن فئة البيانات

يحتوى كائن Dataset على وصف للبيانات. ويمكن استخدام هذا الكائن لإضافة جداول إلى تقرير Crystal. ولإضافة جداول من كائن فئة البيانات، نستخدم خبير قاعدة البيانات (Database Expert) فى مصمم Crystal Reports. ويتم الوصول إلى Database Expert عند تكوين تقرير جديد باستخدام Report Expert. وللوصول إلى Database Expert من خلال تقرير تم تكوينه من قبل، ننقر بزر الماوس الأيمن على مصمم التقارير، نشير إلى Database، ثم ننقر Add/Remove Database. لإجراء اتصال بين تقرير وبين كائن فئة بيانات:

١. نوسع مجلد Project Data فى خبير قاعدة البيانات.

٢. نوسع مجلد ADO.NET Datasets.

٣. نختار كائن Dataset من بين كائنات المشروع.

٤. نختار الجدول الذى نريد إضافته إلى التقرير من جانب Available data sources ، ثم ننقر Insert Table لإضافة الجدول إلى جانب Tables in report. يترتب على ذلك احتواء التقرير على وصف للجداول والحقول وليس على بيانات فعلية ، لأن كائن فئة البيانات يحتوى فقط على وصف البيانات وليس البيانات الفعلية.

٥. لكى يقوم التقرير بعرض البيانات الفعلية، يجب تأهيل كائن dataset بالبيانات قبل ربط التقرير مع متحكم Viewer.

تأهيل فئة البيانات وعرض التقرير

بعد تكوين ملف التقرير وملف فئة البيانات، نحتاج إلى تأهيل فئة البيانات بالبيانات ثم عرض هذه البيانات بالتقرير فى وقت التشغيل. البرنامج التالى يقدم مثالا على تنفيذ التقارير التى تستخدم فئة البيانات. ويستخدم مورد بيانات SQL Server Data Provider:

```
Imports CrystalDecisions.CrystalReports.Engine
Imports System.Data
Imports System.Data.SqlClient
Public Class Form1
   Inherits System.Windows.Forms.Form
```

```
Dim rpt As CrystalReport1
Dim dts As DataSet
Dim cnn As SqlConnection
Dim adp As SqlDataAdapter
Dim cnnstring As String
Dim Sqlstring As String

#Region " Windows Form Designer generated code "

Public Sub New()
    MyBase.New()

    InitializeComponent()

    cnnstring = ""
    cnnstring += "Server=FHOME1; Database=Northwind;"
    cnnstring += "User ID=sa; Password=fnasr;"
    cnn = New SqlConnection(cnnstring)
    Sqlstring = "SELECT * FROM Shippers"
    adp = New SqlDataAdapter(Sqlstring, cnn)
    dts = New DataSet()
    rpt = New CrystalReport1()
    adp.Fill(dts, "Shippers")
    rpt.SetDataSource(dts)
    CrystalReportViewer1.ReportSource = rpt
End Sub

Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    If disposing Then
        If Not (components Is Nothing) Then
            components.Dispose()
        End If
    End If
    MyBase.Dispose(disposing)
End Sub

Private components As System.ComponentModel.IContainer

Friend WithEvents CrystalReportViewer1 As
 CrystalDecisions.Windows.Forms.CrystalReportViewer
<System.Diagnostics.DebuggerStepThrough()> Private Sub
```

```
    InitializeComponent()
    Me.CrystalReportViewer1 = New
    CrystalDecisions.Windows.Forms.CrystalReportViewer()
      Me.SuspendLayout()
      Me.CrystalReportViewer1.ActiveViewIndex = -1
      Me.CrystalReportViewer1.Dock = System.Windows.Forms.DockStyle.Fill
      Me.CrystalReportViewer1.Name = "CrystalReportViewer1"
      Me.CrystalReportViewer1.ReportSource = Nothing
      Me.CrystalReportViewer1.Size = New System.Drawing.Size(292, 266)
      Me.CrystalReportViewer1.TabIndex = 0

      Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
      Me.ClientSize = New System.Drawing.Size(292, 266)
      Me.Controls.AddRange(New System.Windows.Forms.Control()
     {Me.CrystalReportViewer1})
      Me.Name = "Form1"
      Me.Text = "Form1"
      Me.ResumeLayout(False)

   End Sub
#End Region
End Class
```

فيما يلى تحليل للكود الذى يحتوى علية البرنامج السابق:

١. فى بداية وحدة الكود، نضيف مناطق الأسماء التالية:

```
Imports CrystalDecisions.CrystalReports.Engine
Imports System.Data
Imports System.Data.OleDb
```

٢. بعد عبارة الإعلان عن تصنيف النموذج، نعلن عن المتغيرات التالية:

```
Dim rpt As CrystalReport1
Dim dts As DataSet
Dim cnn As SqlConnection
Dim adp As SqlDataAdapter
Dim cnnstring As String
Dim Sqlstring As String
```

٣. فى وسيلة New، نقوم بتكوين سلسلة الاتصال:

```
cnnstring = ""
```

```
cnnstring += "Server=FHOME1; Database=Northwind;"
cnnstring += "User ID=sa; Password=fnasr;"
```

٤. الكود التالى يقوم ببناء كائن الاتصال مع قاعدة البيانات:

```
cnn = New SqlConnection(cnnstring)
```

٥. الكود التالى يقوم بتكوين الاستعلام المرسل إلى قاعدة البيانات:

```
Sqlstring = "SELECT * FROM Shippers"
```

٦. تكوين موفق البيانات اللازم لتأهيل فئة البيانات:

```
adp = New SqlDataAdapter(Sqlstring, cnn)
```

٧. تكوين كائن فئة بيانات من تصنيف فئة البيانات السابق تكوينة:

```
dts = New DataSet()
```

٨. تكوين كائن تقرير من تصنيف التقرير السابق تكوينة:

```
rpt = New CrystalReport1()
```

٩. تأهيل فئة البيانات وعرض التقرير فى متحكم مشاهدة التقارير بنماذج الويندوز:

```
adp.Fill(dts, "Shippers")
rpt.SetDataSource(dts)
CrystalReportViewer1.ReportSource = rpt
```

إضافة التقارير إلى مشروعات التطبيقات

يمكن الاختيار بين إضافة التقارير إلى التطبيقات باستخدام مكون التقارير (Report Component)، أو إضافة التقارير بدون استخدام مكون التقارير. فى حالة إضافة التقارير بدون استخدام مكون التقارير، يمكن إضافة هذه التقارير مباشرة إلى مشروع التطبيق بحيث تصبح جزءا من الكود الناتج عن برنامج الترجمة (The Compiler)، كما يمكن عدم إضافة التقارير إلى مشروعات التطبيقات والاكتفاء باستخدام المسار المؤدى إلى ملفات تلك التقارير. وفى حالة استخدام مكون التقارير لإضافة التقارير إلى التطبيقات، يمكن استخدام مكون تقرير نوعى أو مكون تقرير غير نوعى. يستخدم مكون التقرير النوعي للربط مع تقرير محدد فى وقت التصميم. بينما يستخدم مكون التقرير غير النوعي للربط مع أحد التقارير فى وقت التشغيل.

الإضافة المباشرة للتقارير

تعتبر التقارير التى يتم إضافتها إلى مشروعات Visual Basic من التقارير النوعية. وينتج

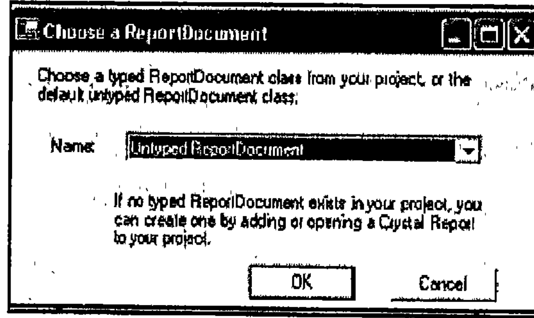
عن إضافتها تكوين ملف مصدر للتقرير يحتوى على تعريف التصنيف الخاص بهذا التقرير. ويرث هذا التصنيف من تصنيف ReportClass الموجودة فى النظام. ويبين ملف تصنيف التقرير الأقسام التى يتكون منها هذا التقرير. ولا يجب أن يقوم المبرمجين بتغيير هذه الأقسام. ويوجد هذا الملف ضمن الملفات المختفية فى مربع Solution Explorer، ويؤدى النقر على أيقونة Show All Files إلى ظهوره تحت ملف امتدادة rpt يمثل تقرير Crystal Report. وتتم إضافة ملفات التقارير إلى المشروع عند تكوين تقرير جديد، كما سبق بيانه. ويمكن أيضا إضافة التقارير الموجودة إلى المشروع، باتباع الخطوات التالية:

١. فى مربع Solution Explorer، ننقر بزر الماوس الأيمن على اسم المشروع، نشير إلى Add ثم نختار Add Existing Item من القائمة المختصرة.

٢. فى مربع حوار Add Existing Item، نختار All Files تحت Files of type. نحدد بعد ذلك الموقع واسم التقرير الذى نريد إضافته.

إضافة التقارير من خلال مكونات التقارير غير النوعية

يمكن اختيار إدراج أحد التقارير فى نموذج ويندوز عن طريق إضافة مكون تقارير غير نوعى (Untyped Report Component) إلى النموذج أولا، ثم تحميل المكون بملف التقرير بعد ذلك. يتبع مكون Untyped Report Component تصنيف ReportDocument. ولا يتم تكوين تصنيف خاص به فى وقت التصميم بسبب ربطة مع التقرير فى وقت التشغيل.



شكل رقم ٢٣

لإضافة مكون تقرير غير نوعى إلى نموذج، نطبق الخطوات التالية:

١. نسحب مكون Report Document من صفحة ملصق Components بمربع ToolBox إلى النموذج.

٢. فى مربع حوار Choose a ReportDocument المبين بالشكل رقم (٢٣)، نختار تصنيف Untyped ReportDocument. يترتب على ذلك إضافة مكون Untyped ReportDocument إلى مربع المكونات اسفل مصمم النماذج. كما يتم إضافة كائن إلى ملف وحدة الكود الخاصة بالنموذج، باستخدام الكود التالى:

```
Protected WithEvents reportDocument1 As _
CrystalDecisions.CrystalReports.Engine.ReportDocument
```

ولتحميل مكون تقرير غير نوعى بأحد التقارير، نستدعى وسيلة Load الخاصة بتصنيف ReportDocument. على سبيل المثال، لتحميل مكون التقرير غير النوعي المسمى ReportDocument1، نستخدم الكود التالى:

```
reportDocument1.Load ("c:\My Report.rpt")
```

بعد تحميل ملف طباعة فى مكون Untyped ReportDocumentيمكن ربط هذا المكون مع متحكم CrystalReportViewer.

إضافة التقارير من خلال مكونات التقارير النوعية

يمكن إضافة تقرير نوعى إلى نموذج ويندوز من خلال مكون تقارير نوعى (Typed Report Component). يسمح لنا ذلك بضبط خيارات الطباعة الخاصة بالمكون فى نافذة Propertiesويمكن أيضا استضافة التقرير عن طريق ربط متحكم مشاهدة التقارير (CrystalReportViewer Control) مع مكون التقرير (Report Component). لإضافة مكون تقارير نوعى إلى المشروع، نطبق الخطوات التالية:

١. نقوم ببناء مشروع.

٢. نسحب مكون ReportDocument من صفحة ملصق Components فى مربع ToolBox إلى النموذج. يترتب على ذلك استدعاء مربع حوار Choose a ReportDocument.

٣. نختار تصنيف التقرير النوعي من بين تصنيفات التقارير التى يعرضها مربع الحوار السابق. وإذا لم يكن بالمشروع تقرير نوعى، يمكن إضافة إحداها عن طريق إضافة تقرير إلى المشروع بالنقر على اسم المشروع فى مربع Solution Explorer ثم اختيار Add New Item، Crystal Report من القائمة المختصرة.

يترتب على الخطوات السابقة، إضافة مكون ReportDocument إلى مربع المكونات اسفل مصمم نماذج الويندوز. وبعد إضافة المكون المذكور إلى النموذج، يتم إضافة كائن مقابل فى وحدة الكود الخاصية بنموذج الويندوز باستخدام الكود التالى:

```
Protected WithEvents my_Report1 As My_Project.My_Report
```

ويجب ربط مكون التقرير الذى تم تكوينه مع متحكم مشاهدة التقارير قبل أن نستطيع رؤية التقرير.

ربط التقارير مع أدوات مشاهدة التقارير

قبل أن نستطيع عرض تقرير فى متحكم مشاهدة التقارير (CrystalReportViewer Control)، يجب ربط كائن التقرير مع ذلك المتحكم. يتم ذلك عن طريق ضبط خاصية ReportSource فى متحكم Viewer. ويمكن ربط التقرير فى وقت التصميم، كما يمكن أيضا ربط تقرير فى وقت التشغيل لدعم عرض تقارير مختلفة على أساس التفاعل مع المستخدم أو متحكم أخر على نموذج الويندوز.

خطوات إضافة متحكم مشاهدة التقارير إلى النموذج

١. نفتح مربع الأدوات ثم نسحب متحكم CrystalReportViewer إلى النموذج.
٢. نغير حجم ونحرك المتحكم المذكور كما نرغب عن طريق السحب والإسقاط (Drag and Drop).
٣. عند تشغيل التطبيق، سوف يتم عرض التقرير فى متحكم CrystalReportViewer.
٤. وكما هو الحال مع أى متحكم فى مربع الأدوات، يمكن إضافة كود إلى هذا المتحكم عن طريق النقر المزدوج علية لعرض محرر الكود.

ضبط خصائص متحكم مشاهدة التقارير بنماذج الويندوز

يمكن ضبط القيم الابتدائية لخصائص متحكم CrystalReportViewer في وقت التصميم. ويمكن أيضا ضبط أو تغيير هذه الخصائص في وقت التشغيل باستخدام الكود. تشمل خصائص متحكم Viewer الخصائص المتعلقة بالمتحكم والخصائص العامة التي تنطبق على جميع أدوات التحكم على النموذج.

لضبط خصائص متحكم مشاهدة التقارير في وقت التصميم:

١. نختار متحكم CrystalReportViewer على نموذج الويندوز ثم نعرض نافذة Properties.

٢. في نافذة Properties، نغير قيم الخصائص التي نريد تغييرها.

لتنفيذ الضبط أثناء وقت التشغيل، ندخل الكود المناسب لضبط الخصائص في وحدة الكود الخاصة بنموذج الويندوز.

ربط التقارير غير المضافة إلى المشروع

عندما توجد التقارير في صورة ملفات بالكمبيوتر غير مضافة إلى المشروع محل التصميم، توجد ثلاثة وسائل للربط: الربط باستخدام اسم ملف التقرير، الربط باستخدام كائن يمثل التقرير، والربط باستخدام مكون تقرير غير نوعي(Untyped ReportDocument)

ربط التقرير باستخدام اسم ملف التقرير

- لإجراء الربط باستخدام نافذة Properties، نختار متحكم Viewer على نموذج الويندوز ثم نضبط خاصية ReportSource على اسم ملف التقرير.
- لإجراء الربط باستخدام الكود، نفتح محرر الكود ثم نضيف كود يماثل الكود التالي مع استخدام اسم الملف الصحيح الخاص بالتقرير:

```
CrystalReportViewer1.ReportSource = "C:\\Reports\\My Report.rpt"
```

ربط التقرير غير المضاف باستخدام كائن تقرير

١. نفتح محرر الكود بالنقر المزدوج على النموذج الأساسي في.

٢. في وحدة الكود الخاصة بالنموذج، نضيف نطاق الأسماء التالي:

```
Imports CrystalDecisions.CrystalReports.Engine
```

٣. نعلن عن متغير عام من نوع ReportDocument على مستوى وحدة الكود:

```
Public WithEvents oRpt As ReportDocument
```

٤. في وسيلة New()، نستخدم متغير كائن التقرير في تحميل التقرير، ثم نربط الكائن بعد ذلك مع متحكم Viewer في نموذج الويندوز.

```
Public Sub New ()
  MyBase.New ()
  InitializeComponent ()
  oRpt = New ReportDocument ()
  oRpt.Load ("C:\\Reports\\My Report.rpt")
  CrystalReportViewer1.ReportSource = oRpt
End Sub
```

الربط باستخدام مكون تقرير غير نوعى

نفترض إضافة مكون تقرير غير نوعى باسم reportDocument1 إلى نموذج ويندوز، ثم تحميل المكون بالملف C:\Reports\MyReport.rpt. يمكننا ربط مكون التقرير غير النوعي مع متحكم Viewer على النموذج عن طريق ضبط خاصية ReportSource باستخدام الكود :

```
CrystalReportViewer1.ReportSource = reportDocument1
```

ربط التقارير المضافة إلى المشروع

يمكن ربط التقارير المضافة إلى المشروع مع متحكم CrystalReportViewer، عن طريق استخدام مكون تقرير نوعى أو عن طريق استخدام كائن من تصنيف التقرير.

ربط التقرير باستخدام كائن تقرير

عند إضافة أحد التقارير إلى مشروع تطبيق ويندوز، يتم تلقائيا إضافة ملف تصنيف امتدادة (.rpt)، إلى المشروع خاص بهذا التقرير. ويمكن تكوين كائن من هذا التصنيف وربطة مع متحكم Viewer على نموذج الويندوز عن طريق تخصيص كائن التقرير لخاصية ReportSource باستخدام الكود التالى:

```
CrystalReportViewer1.ReportSource = new My_Report ()
```

ربط التقرير باستخدام مكون تقرير نوعى

يمكن ربط التقرير المضاف إلى المشروع مع متحكم Viewer من خلال مكون تقرير نوعى،

باستخدام نافذة الخصائص أو باستخدام الكود.

- عند الربط باستخدام نافذة Properties، نختار متحكم Viewer على نموذج الويندوز ثم نضبط خاصية ReportSource على اسم مكون التقرير النوعي عن طريق الاختيار من القائمة المنسدلة على يمين هذه الخاصية.
- ويمكن القيام بعملية الربط باستخدام الكود التالى فى محرر الكود:

```
CrystalReportViewer1.ReportSource = my_Report1
```

التعامل مع كائنات التقرير

يستخدم مصمم Crystal Reports مدخل سحب وإسقاط مماثل للمستخدم فى Visual Studio .NET. ويقوم نظام التعامل مع كائنات الحقول على أساس سحب أحد هذه الكائنات، مثل حقل بيانات أو كائن نص، من مربع Field Explorer إلى المصمم ثم استخدام نافذة Properties أو استخدام قائمة مختصرة لصياغة ذلك الكائن. وبالنسبة للكائنات التى لا توجد فى مربع Field Explorer، نستخدم القائمة المختصرة فى إضافتها إلى التقرير. نحصل على القائمة المختصرة بالنقر على سطح مصمم التقرير بزر الماوس الأيمن.

كائنات التقرير

هناك الكثير من كائنات التقارير التى يمكن إضافتها وصياغتها طبقا لاحتياجاتنا. يشمل ذلك الكائنات التالية:

- حقول قاعدة البيانات (Database fields).
- حقول صياغة (Formula fields).
- حقول معاملات (Parameter fields).
- حقول مجموعات (Group Name fields).
- حقول تعبيرات لغة الاستعلام (SQL Expression Fields)
- حقول الإجمالي المتحرك (Running Total fields).
- الحقول الإجمالية (Summary fields).
- الرسوم البيانية (Charts).

- التقارير الفرعية (Subreports).

أدراج وتحريك الحقول والكائنات باستخدام القائمة المختصرة

١. ننقر بزر الماوس الأيمن على مصمم التقرير، نشير إلى Insert ونختار حقل أو كائن من القائمة المختصرة.

٢. فى حالة عرض مربع حوار، يتم إدخال المعلومات المطلوبة.

٣. نسحب إطار الكائن إلى القسم المطلوب على التقرير.

اختيار كائنات الحقول

لتنفيذ أحد التصرفات على كائنات الحقول، يجب أولا اختيار هذه الكائنات. وعند اختيار أحد الحقول، يظهر حول الكائن إطار وبه مقابض على الجوانب الأربعة. تشير هذه المقابض إلى أن الحقل قد تم اختيارة. لاختيار أحد الحقول الموجودة على التقرير:

١. نضع مؤشر الماوس بداخل إطار الكائن ثم ننقر مرة واحدة. يترتب على ذلك ظهور المقابض التى تشير إلى اختيار هذا الكائن.

٢. لإلغاء اختيار أحد الحقول، نحرك مؤشر الماوس بعيدا عن الحقل ثم ننقر على أى مكان بالنافذة. يترتب على ذلك اختفاء المقابض الموجودة حول الكائن.

تغيير حجم كائنات الحقول

١. نختار الحقل الذى نريد تغيير حجمة. ويمكن تغيير حجم حقل وعنوان الحقل معا بالضغط على مفتاح CTRL والنقر بزر الماوس فى نفس الوقت لاختيار الحقل وعنوانه.

٢. نشير إلى أحد مقابض تغيير الحجم على جوانب الإطار.

٣. وعندما يأخذ مؤشر الماوس شكل سهم تغيير الحجم، نسحب المقبض لتغيير الحجم.

حذف الحقول

١. نختار الحقل المطلوب حذفة. ويمكن اختيار الحقل وعنوانه بالضغط على CTRL

والنقر معا.

٢. نضغط مفتاح Delete.

عرض أسماء الحقول

عند إضافة حقول إلى التقرير، يمكن اختيار نوع معلومات الحقل التى نريد عرضها أثناء تصميم التقرير. يكن اختيار عرض اسم الحقل، عرض اسم الكائن، أو عرض صيغة الحقل. لعرض أسماء الحقول أو تغيير خيارات الضبط:

١. ننقر بزر الماوس الأيمن على مصمم التقرير، نشير إلى Designer ثم ننقر Default Settings.

٢. فى مربع حوار Default Settings، نختار مربع Show Field Names، أو أى خيار أخر.

٣. ننقر Ok.

إضافة عناوين للحقول

عند إضافة حقل من قاعدة بيانات إلى تقرير، يقوم البرنامج تلقائيا بإضافة عنوان للحقل بناءا على اسم ذلك الحقل فى قاعدة البيانات. ولإضافة عنوان أحد الحقول يدويا:

١. ننقر بزر الماوس الأيمن على مصمم التقارير، نشير إلى Insert ثم ننقر Text Object.

٢. نضع كائن النص فى قسم Page Header ثم نصفف هذا الكائن فوق كائن الحقل.

٣. ننقر نقرا مزدوجا على كائن النص لوضع مؤشر الإدراج داخل إطار الكائن.

٤. نطبع الاسم الذى نريد استخدامه.

٥. لصياغة العنوان، ننقر بزر الماوس الأيمن على كائن النص ثم نختار Format.

إدراج حقل قاعدة بيانات فى كائن نص

١. نضع كائن نص (Text Object) داخل التقرير.

٢. نقوم بالنقر المزدوج على كائن النص ثم نطبع النص المطلوب ظهوره قبل حقل البيانات.

٣. فى مربع Field Explorer، نحدد الحقل الذى نريد إضافته إلى كائن النص.

٤. نسحب حقل قاعدة البيانات من مربع Field Explorer إلى كائن النص. ويجب ضبط مؤشر الإدراج فى داخل كائن النص على المكان الذى نريد وضع حقل قاعدة البيانات به.

دوران كائنات الحقول

تحت ملصق Common فى مربع حوار Format Editor، يمكننا استخدام خيارات Text rotation لتصفيف كائنات الحقول والنصوص رأسيا على التقرير. ويمكننا تحريك النص ٩٠ أو ٢٧٠ درجة.

لتدوير كائن نص :

١. ننقر بزر الماوس الأيمن على كائن تقرير ونختار Format.

٢. تحت ملصق Common، نختار درجة الدوران فى حقل Text Rotation.

منع بتر النصوص

عند وضع كائن نص على تقرير، يتم تمثيلة بإطار على وجه مصمم التقارير. يعتمد ارتفاع هذا الإطار على ارتفاع بنط الحروف المستخدم. بينما يتحدد العرض على أساس الكائن الذى نعمل معه. وبغض النظر عن عرض الكائن، سواء كان العرض الإفتراضى أو عرض تم تغييره بواسطة المستخدم، يمكن أن نواجه مشكلة زيادة طول النص المطبوع على عرض كائن النص مما يترتب علية بتر النص عند الطباعة. وقد يبدو التقرير فى صورة جيدة على الجهاز المستخدم فى تصميمه، إلا أنة عند الطباعة باستخدام محرك طابعة مختلف، يتمدد طول النص ويبقى إطار الكائن ثابتا مما يؤدى إلى بتر النص.

- بالنسبة لحقول قاعدة البيانات غير الحقول من نوع memo، يتقرر العرض على أساس عرض الحقل فى قاعدة البيانات، وعلى أساس متوسط عرض الحرف الذى يحدده طاقم الحروف المستخدم وحجم الحرف. على سبيل المثال، نفترض وجود حقل قاعدة بيانات يسمى (Customer.LASTNAME). وأن قاعدة البيانات تحدد هذا الحقل على أنة حقل نص بطول ٣٥ حرف. عند وضع هذا الحقل على

التقرير، يساوى العرض ٣٥ مرة متوسط عرض وحجم البنط المستخدم فى صياغة حقل قاعدة البيانات. ويعتبر ذلك هو العرض الإبتدائى الذى يمكن تغييره فيما بعد.

- بالنسبة لكائنات النصوص، يكون العرض الإفتراضى مساويا ١٧ مرة من متوسط عرض الحرف. وتختلف كائنات النصوص عن حقول قاعدة البيانات فى أنها تتمدد تلقائيا عند إدخال نص أو حقل بيانات فى الكائن. ومثل كائنات النصوص الأخرى ، يمكن تغيير الحجم يدويا.
- يختلف العرض الإفتراضى لحقول الأرقام على أساس نوع البيانات الرقمية التى تحتوى عليها (double، single، integer، long integer، و byte).

لمنع بتر النص داخل كائن:

١. نـنقر بزر الماوس الأيمن على الكائن الذى نريد صياغته ثم ننقر Format فى القائمة المختصرة.

٢. فـى مـربع حـوار Format Editor، ننقر على ملصق Common ونختار مربع اختيار Can Grow.

٣. ننقر OK لحفظ التغيير. يترتب على ذلك صياغة الكائن للطباعة على سطور متعددة. فإذا كانت طباعة النص أكبر من عرض الكائن ، فإن النص يجرى استكماله على سطور إضافية.

منع تداخل النصوص

يجب تجنب تصميم التقارير حيث تكون المسافة بين الكائنات ضيقة جدا. بل يجب ترك مسافة للنمو الناتج عن توسيع عرض الكائن بنسبة ٥٪ تقريبا، وإذا لم يكن ذلك ميسرا، يجب إنقاص حجم البنط.

إذا كان محرك الطابعة يمدد أو يقلص مسافات النص، يمكن أن تختلف عملية التفاف الكلمات بتغيير عدد السطور الضرورية لطباعة الكائن لكى يتم التوافق مع النمو أو الانكماش. ويمكن مواجهة مشكلة عند وضع كائنات أخرى مباشرة تحت كائنات نصوص

متعددة السطور. وعلى خلاف كائنات النصوص ذات السطر الواحد، لا يتوفر خيار توسيع كائن النص فى كائنات النصوص متعددة السطور. ولهذا، يجب وضع كائنات النصوص متعددة السطور فى أسفل القسم. فإذا تطلب الكائن سطورا أكثر للطباعة فإن القسم يتمدد إلى أسفل ليتوافق مع النمو ولا يؤثر ذلك على الكائنات الأخرى.

زيادة المساحات الفارغة بين الأقسام

ارتفاع القسم بالنسبة للكائنات التى تقع بداخلة يؤثر على كمية المساحات البيضاء بين السطور فى التقرير. لإضافة مساحة بيضاء إضافية بين الصفوف فى تقرير، نحرك المؤشر على الحد الأسفل فى القسم. يترتب على ذلك، تغيير شكل المؤشر إلى مؤشر تغيير الحجم (Resizing Cursor). بعد ظهور مؤشر تغيير الحجم ، نسحب حد القسم إلى أسفل لإضافة مساحة بيضاء إضافية.

منع بتر الأرقام

إذا كانت قيمة رقمية أو قيمة عملة أكبر من الحقل الذى يحتوى عليها، فإن الوضع الطبيعى هو بتر أو قص هذه القيمة. على سبيل المثال ، قيمة مثل ١٠٠٠٠٠٠٠٠ يمكن أن تظهر على التقرير ١٠٠٠. يمكن أن يؤدى ذلك إلى حدوث ارتباك عند قراءة التقرير. وعند إزالة خيار Allow Field Clipping، سوف يتم تمثيل القيم الرقمية والنقدية التى تتجاوز حجم الحقل بالعلامة الرقمية (######)، للإشارة إلى أن الحقل صغير بالنسبة للرقم.

للسماح باستخدام علامات تجاوز الحقل:

١. ننقر بزر الماوس الأيمن على حقل العملة أو الرقم الذى نريد صياغته ثم ننقر Format. يترتب على ذلك ظهور مربع حوار Format Editor مع عرض صفحة ملصق Number.

٢. ننقر على زر Customize. يؤدى ذلك إلى ظهور مربع حوار Custom Styles مع فتح ملصق Number.

٣. للسماح بتمثيل تجاوز حجم الحقل، نزيل إشارة مربع فحص Allow Field Clipping. وهناك خيار أخر متاح لنا لإدخال صيغة فى Formula Editor، هو زر

Format. فى مربع حوار Format Formula Editor، يمكننا تحديد أن قص الحقل سوف يتم حجبة فى حالة مقابلة شروط معينة.

٤. ننقر OK لحفظ التغييرات.

٥. عند حجب إمكانية قص الحقل، فإن أى قيمة رقمية أو نقدية أكبر من الحقل الذى يحتويها، سوف يتم تمثيلها بعلامات الرقم (######).

التحكم فى بيانات التقرير

يتناول هذا القسم عمليات التحكم فى البيانات التى يحتوى عليها التقرير وطريقة عرضها. تشمل هذه العمليات: ترشيح البيانات، ضبط المعاملات، تكوين المجموعات وتلخيصها، فرز البيانات، تكوين الإجماليات الفرعية والمتحركة، استخدام الصيغ، وتنسيق البيانات.

ترشيح البيانات

يبين لنا هذا القسم كيفية ترشيح البيانات التى نريد وضعها فى التقرير. ويتم ترشيح البيانات باستخدام صيغ اختيار السجلات واستخدام المعاملات. على سبيل المثال، باستخدام أدوات اختيار السجلات، يمكننا جعل التقرير يشتمل على مجموعة معينة من العملاء، نطاق محدد من أرقام الحسابات، أو نطاق تواريخ خاص.

اختيار السجلات

عند اختيار سجل لعرضة على تقرير، فإن قيم الحقول فى كل السجلات بالجدول المستخدم يجرى طباعتها افتراضيا. غير أنة فى حالات كثيرة قد لا نريد وضع جميع هذه القيم فى التقرير، ولكن فقط طباعة فئة فرعية من هذه القيم. لتوفير هذه الإمكانية، يحتوى Crystal Reports على لغة صياغة معقدة يمكن استخدامها افتراضيا لتحديد أى نوع من أنواع اختيار السجلات. كما يحتوى على أداة خبير الاختيار (Select Expert) التى تقوم بقيادة المستخدمين فى عملية تكوين الاختيارات المختلفة. على هذا الأساس يمكن التفريق بين طريقتين لاختيار السجلات فى Crystal Reports:

- استخدام Select Expert لمعالجة أساسيات اختيار السجلات.

- اختيار السجلات باستخدام الصيغ.

تحديد الحقول التى نستخدمها فى اختيار السجلات

عند اختيار سجلات، فإن ذلك يعني بناء التقرير على سجلات تتوافق مع الشروط التى قمنا بتحديدها. وتعتمد هذه الشروط على نوع البيانات التى نريد عرضها بالتقرير. نفترض على سبيل المثال، أننا نريد تكوين تقرير يعرض بيانات عن منطقة معينة. التحدي الذى نواجهة فى هذه الحالة هو العثور على أحسن طريقة للتعرف على السجلات التى تخص تلك المنطقة.

- إذا كان الجدول المستخدم فى التقرير يحتوى على حقل يمثل المنطقة، يمكننا توجيه البرنامج نحو اختيار السجلات التى تحتوى على اسم المنطقة المستهدفة فى الحقل المذكور فقط.
- إذا كان الجدول لا يحتوى على حقل يمثل المنطقة، نبحث عن طريقة أخرى لتحديد المنطقة.
- إذا كان الجدول يحتوى على حقل خاص بالرمز البريدي، يمكن اختيار المنطقة المستهدفة بناءا على نطاق الرموز البريدية التى تتبع هذه المنطقة.
- إذا كان الجدول يحتوى على حقل خاص بكود المنطقة، يمكن اختيار المنطقة بناءا على قيمة هذا الحقل.

دفع معيار اختيار السجلات إلى خادم قاعدة البيانات

تسمح المحركات التى يوفرها Crystal Reports للتعامل مع مصادر بيانات SQL بدفع معايير اختيار السجلات إلى خادم البيانات. وعند تحديد صيغة لاختيار السجلات فى تقرير يعتمد على مصدر بيانات SQL، يقوم Crystal Reports بتحليل هذه الصيغة، تكوين استعلام SQL على أساسها وتمرير الاستعلام إلى خادم SQL. يتم بعد ذلك تنفيذ هذا الاستعلام على مرحلتين :

- فى المرحلة الأولى من اختيار السجلات، يقوم خادم قاعدة البيانات بمعالجة الاستعلام وإعادة فئة من السجلات إلى Crystal Reports.

- فى المرحلة الثانية، يقوم Crystal Reports بتقييم صيغة اختيار السجلات محليا بالنسبة لفئة السجلات العائدة من خادم قاعدة البيانات ويتم اختيار السجلات النهائية.

الأنواع التالية من معايير اختيارات السجلات، يمكن دفعها إلى خادم SQL :

- الاختيار باستخدام الحقول المفهرسة وغير المفهرسة.
- الاختيار باستخدام استعلامات SQL مع عبارات AND و OR.
- حقول تعبيرات SQL التى تنفذ صيغة العمليات الحسابية الخاصة باختيار السجل.

ويجب ملاحظة أن هناك صيغ لايمكن تنفيذها على الخادم. يوضح المثال التالى مزايا كتابة صيغ اختيار السجلات التى يمكن تنفيذها على خادم قاعدة البيانات. نفترض أن جدول Orders فى قاعدة البيانات الافتراضية Xtreme به ٢٠٠١ سجل، منها ١٦٩ سجل يقع تاريخها قبل سنة ١٩٩٧. ونفترض أننا نريد إعداد تقرير عن تلك السجلات فقط. يمكننا استخدام صيغة الاختيار التالية:

Year ({Orders.Order Date}) < 1997

استعلام SQL الناتج عن هذه الصيغة، سوف يقوم بإرسال كل السجلات البالغ عددها ٢٠٠١ إلى Crystal Reports، ثم تقوم صيغة الاختيار المذكورة على الكمبيوتر العميل بتخفيض عدد السجلات التى يتم اختيارها إلى ١٦٩ سجل. يرجع السبب فى ذلك إلى أن الاستعلام الذى تم تكوينه لا يحتوى على فقرة WHERE لأن دالة Year( ) لايمكن تنفيذها على خادم البيانات.

من ناحية أخرى، يمكن استخدام صيغة الاختيار التالية:

{Orders.Order Date} < #Jan 1, 1997#

يمكن تنفيذ الصيغة الثانية على خادم قاعدة البيانات، وبالتالى ينتج عن الاستعلام الذى تكونه هذه الصيغة إعادة ١٦٩ سجل فقط إلى Crystal Reports.

استخدام *Select Expert* فى تكوين معايير الترشيح

يجعل Select Expert من السهل اختيار السجلات التى نريد استخدامها فى تقاريرنا.

وعند العمل مع هذه الأداة، نختار الحقل الذى نريد تطبيق شروط الاختيار علية ثم نحدد هذه الشروط.

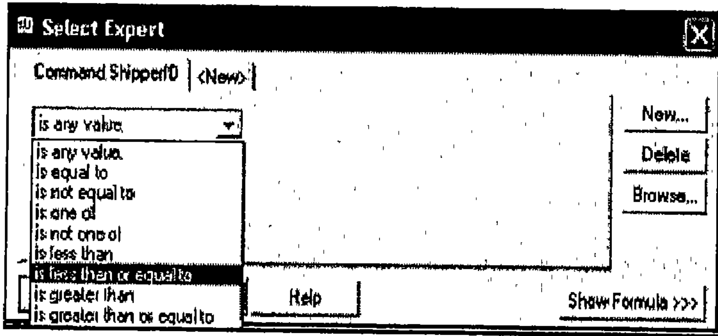
لإعداد صيغة اختيار سجل بواسطة Select Expert:

١. نـنقر بزر الماوس الأيمن على مصمم التقارير، نشير إلى Report ثم ننقر على Select Expert. يترتب على ذلك عرض مربع حوار Choose Field.

٢. فى مربع حوار Choose Field، نركز الضوء على الحقل الذى نريد إستخدامة فى عملية الاختيار ثم ننقر على زر OK. ويمكن اختيار أكثر من حقل بالنقر على ملصق New، ثم اختيار الحقل التالى.

٣. فى مربع حوار Select Expert المبين بالشكل رقم (٢٤)، نستخدم القوائم المنسدلة لإدخال معيار الاختيار بالنسبة للحقل السابق اختيارة فى الخطوة السابقة.

٤. ننقر على زر OK عند الانتهاء.



شكل رقم ٢٤

إعداد صيغة الاختيار بواسطة المستخدم

١. ننقر بزر الماوس الأيمن على مصمم التقارير، نشير إلى Report، نختار Edit Selection Formula ثم ننقر Records. يترتب على ذلك عرض مربع حوار Record Selection Formula Editor المبين بالشكل رقم (٢٥).

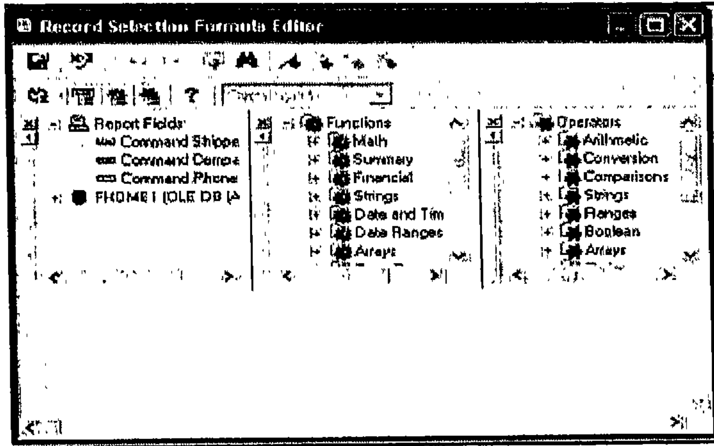
٢. فى مربع حوار Record Selection Formula Editor، ندخل الصيغة فى مربع

الإدخال بأسفل النافذة أو الاختيار من أشجار المكونات (Field Tree، Function Tree، Operator Tree).

٣. ننقر زر Check للتعرف على أى خطأ فى الصيغة.

٤. نصحح أخطاء الصيغة إن وجدت.

٥. ننقر زر Save and Close بعد التحقق من صحة الصيغة.



شكل رقم ٢٥

قوالب صيغ اختيار السجلات

يمكن استخدام أمثلة الصيغ التالية قوالب للمساعدة فى تكوين الصيغ الخاصة بنا باستخدام مربع حوار Selection Formula Editor.

اختيار السجلات التى تبدأ فيها قيمة حقل (file.FIELD) بالحرف "C".

```
{file.FIELD} startswith "C"
```

اختيار السجلات التى لا تبدأ فيها قيمة حقل (file.FIELD) بالحرف "C".

```
not ({file.FIELD} startswith "C")
```

اختيار السجلات التى تكون المفردات من ٣ إلى ٥ فى حقل (file.FIELD) تساوى "999".

```
"999" in {file.FIELD}[3 to 5]
```

اختيار السجلات التى تكون قيمة الحقل (file.FIELD) بها تحتوى على السلسلة "Cycle".

```
"Cycle" in {file.FIELD}
```

اختيار السجلات التى تكون قيمة حقل (file.FIELD) فيها أكبر من ٩٩٩٩٩

```
{file.FIELD} > 99999
```

اختيار السجلات التى تكون قيمة حقل (file.FIELD) فيها أقل من ٩٩٩٩٩

```
{file.FIELD} < 99999
```

اختيار السجلات التى تكون قيمة حقل (file.FIELD) اكبر من ١١١١١ وأقل من ٩٩٩٩٩

```
{file.FIELD} > 11111 and {file.FIELD} < 99999
```

اختيار السجلات التى تكون قيمة التاريخ بها أقل من سنة ١٩٩٩

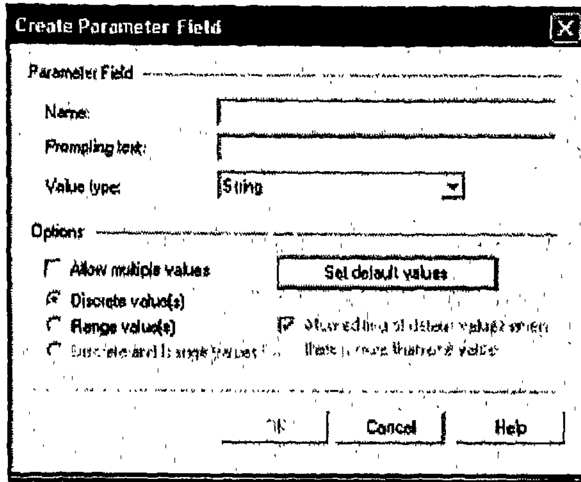
```
Year ({file.DATE}) < 1999
```

اختيار السجلات التى يقع تاريخها بين سنة ١٩٩٢ وسنة ١٩٩٦

```
Year ({file.DATE}) > 1992 and
Year ({file.DATE}) < 1996
```

ضبط المعاملات

تقوم المعاملات (Parameters) بحث مستخدم التقرير على إدخال معلومات. ويمكن النظر إلى المعامل على أنة سؤال يجب على المستخدم الإجابة علية قبل إنتاج التقرير. وتقرر المعلومات التى يدخلها المستخدم أو الطريقة التى يجبب بها، ما سوف يظهر على التقرير. على سبيل المثال، فى تقرير يستخدمه مندوب مبيعات، من الممكن أن يكون هناك معامل يطلب من المستخدم اختيار المنطقة. يترتب على ذلك، طباعة تقرير يحتوى على النتائج الخاصة بمنطقة معينة، بدلا من طباعة نتائج جميع المناطق. ويترتب على استخدام حقول المعاملات فى الصيغ، صيغ الاختيار، وفى التقرير ذاته، تكوين تقرير واحد قابل للتعديل عند الحاجة إلى ذلك. ويمكن أيضا استخدام حقول المعاملات فى التقارير الفرعية.



شكل رقم ٢٦

تصميم حقول المعاملات

١. فى مربع Field Explorer، ننقر بزر الماوس الأيمن على Parameter Fields ثم ننقر New.

٢. فى مربع حوار Create Parameter Field المبين بالشكل رقم (٢٦)، ندخل اسم للمعامل فى مربع حقل Name.

٣. ندخل نص الحث المناسب فى حقل Prompting Text. وهو النص الذى يظهر فى مربع حوار Enter Parameter Value عند تجديد بيانات التقرير (Refreshing).

٤. بندخل نوع القيمة.

٥. ننقر Set Default Values عندما نريد تحديد الخيارات المتاحة أمام المستخدم.

٦. فى مربع حوار Set Default Values، نحدد الجدول والحقل بالنسبة لقيمة المعامل.

٧. ننقر على >> لتحريك أى قيمة إلى منطقة القيم الافتراضية. يؤدى ذلك إلى تمكين المستخدم من اختيار أى قيمة فى منطقة القيم الافتراضية.

٨. ننقر OK. يترتب على ذلك ظهور مربع حوار Create Parameter Field. ننقر Ok

مرة أخرى.

٩. نسحب المعامل ونضعة على التقرير.

تحديد نوع وصيغة الإدخال في المعاملات

عند تحديد القيم الافتراضية لحقل معامل من نوع String، يمكننا اختيار قناع إدخال (Edit Mask) فى حقل Edit Mask، بدلا من تحديد نطاق للقيم. ويمكن أن يكون قناع الإدخال أى مجموعة من الرموز المستخدمة لتقييد القيم التى يمكن إدخالها فى حقول المعاملات. وفيما يلى بيان بالحروف المستخدمة فى تكوين أقنعة الإدخال :

- **"A"** يسمح بإدخال الحروف والأرقام ويفرض إدخال رمز فى قيمة المعامل.
- **"a"** يسمح بإدخال الحروف والأرقام ولا يفرض إدخال رمز فى قيمة المعامل.
- **"0"** يسمح بإدخال الأرقام من صفر إلى تسعة ويفرض إدخال رمز فى قيمة المعامل.
- **"9"** يسمح بإدخال الأرقام أو المسافات ولا يفرض إدخال رمز فى قيمة المعامل.
- **"#"** يسمح باد خال رقم ، مسافة ، أو علامة زائد وناقص ، ولا يفرض إدخال رمز فى قيمة المعامل.
- **"L"** يسمح بإدخال حرف [A to Z] ويفرض إدخال رمز فى قيمة المعامل.
- **"?"** يسمح بإدخال حرف ، ولا يفرض إدخال رمز فى قيمة المعامل.
- **"C"** يسمح بإدخال أى رمز أو مسافة ، ولا يفرض إدخال رمز فى قيمة المعامل.
- **", : ; - /"** رموز فاصلة تستخدم للفصل فى قناع الإدخال.
- **"<"** يسبب تحويل الرموز التالية له إلى الحجم الصغير (Lowercase).
- **">"** يسبب تحويل الرموز التالية له إلى الحجم الكبير (Uppercase).
- **"\"** يسبب عرض الحرف التالى لة بدون تغيير فى قيمة المعامل.
- **"Password"** يسمح لنا باستخدام القناع لإدخال كلمات المرور إلى أقسام التقرير المختلفة.

تكوين المجموعات

فى كثير من التقارير، نحتاج إلى تقسيم البيانات إلى مجموعات لكى نجعلها أسهل فى القراءة والتحليل. يبين لنا هذا القسم كيفية تقسيم البيانات فى مجموعات بالتقرير.

لتكوين مجموعات بيانات :

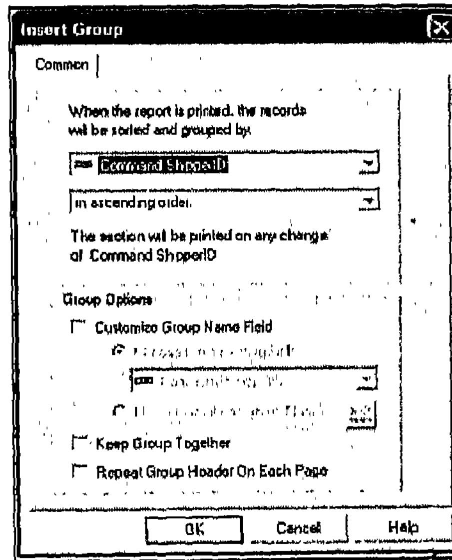
١. ننقر بزر الماوس الأيمن على مصمم التقرير، نشير إلى Insert، ثم ننقر Group.

٢. فى مربع حوار Insert Group المبين بالشكل رقم (٢٧)، نختار من القائمة حقل التقسيم المنسدلة بالقمة.

٣. نختار اتجاه الفرز من القائمة المنسدلة الثانية.

٤. عندما نرغب فى تحديد القيمة المبينة فى مقدمة المجموعة، نختار مربع فحص Customize Group Name Field. من الناحية الافتراضية، سوف يقوم رأس المجموعة بعرض قيمة حقل تقسيم المجموعات. وإذا رغبنا فى عرض قيمة مختلفة، نختار حقل بيانات بديل، أو نكون صيغة.

٥. ننقر OK.



شكل رقم ٢٧

وضع البيانات فى مجموعات هرمية

يوفر لنا Crystal Reports أيضا إمكانية وضع البيانات على التقرير فى مجموعات تبين العلاقات الهرمية. وعند تكوين المجموعات الهرمية، نقوم بفرز المعلومات على أساس العلاقة بين الحقول. على سبيل المثال، عندما نريد إظهار هيكل هرمى لأحد الأقسام، يمكن تقسيم البيانات إلى مجموعات على أساس رقم الموظف (Employee ID) وتحديد الشكل الهرمي باستخدام حقل بيانات المستوى الذى يتبعة الموظف.

لتقسيم البيانات إلى مجموعات هرمية:

١. ننقر بزر الماوس الأيمن على مصمم التقرير، نشير إلى Insert ثم ننقر Group.

٢. فى مربع حوار Insert Group، نختار الحقل الذى يقوم علية التنظيم الهرمي. على سبيل المثال، عندما نريد مشاهدة البناء الهرمي للموظفين فى شركة، نختار حقل بيانات رقم الموظف. وعندما نريد مشاهدة هرم مكاتب مبيعات المناطق، نختار اسم المكتب.

٣. نختار in ascending order.

٤. تلقائيا، تعرض مقدمة المجموعة (Group Header) فى التقرير، قيمة الحقل المستخدم فى تقسيم المجموعات. وعندما نريد إظهار قيمة مختلفة، نضع علامة فى مربع فحص Customize Group Name.

٥. ننقر OK. يترتب على ذلك إضافة المجموعة إلى التقرير.

٦. ننقر بزر الماوس الأيمن على مصمم التقرير، نشير إلى Report، ننقر Hierarchical Grouping Options. يترتب على ذلك عرض مربع حوار Hierarchical Options الموضح بالشكل رقم (٢٨).

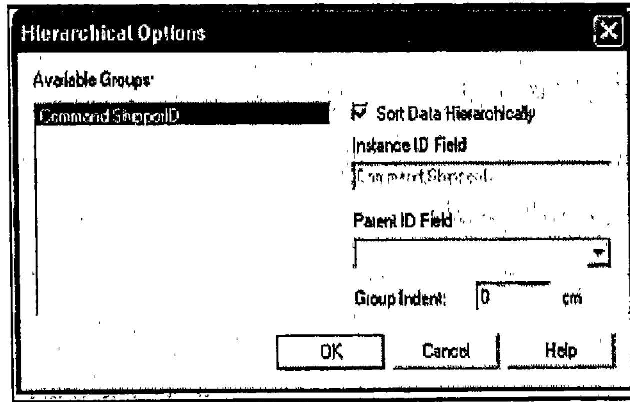
٧. فى قائمة المجموعات المتاحة، نختار المجموعة التى نريد تنظيمها هرميا. فإذا كان التقرير يحتوى على مجموعة واحدة، فإن هذه المجموعة يتم اختيارها تلقائيا فى قائمة المجموعات المتاحة.

٨. نختار مربع الفحص Sort Data Hierarchically.

٩. فى قائمة Parent ID field، نختار الحقل الذى يتبعة حقل Instance ID. على سبيل المثال، بالنسبة لتقرير هرمى خاص بشركة، يمكننا اختيار حقل المدير الذى يرفع الموظف تقارير إلية. ويجب أن يكون كل من حقل Instance ID وحقل Parent ID من نفس نوع البيانات.

١٠. فى حقل Group Indent، ندخل كمية مسافة الزحزحة إلى الداخل لكل مجموعة.

١١. ننقر OK.



شكل رقم ٢٨

تلخيص بيانات المجموعات

أحد الأغراض الأساسية لتقسيم البيانات إلى مجموعات هو إجراء العمليات الحسابية على كل مجموعة من السجلات بدلا من إجرائها على جميع السجلات فى التقرير. وعندما يقوم البرنامج بتلخيص البيانات، فإنه يقوم بفرز البيانات، تقسيمها إلى مجموعات، ثم تلخيص القيم فى كل مجموعة. ويتم القيام بذلك أتوماتيكيا. ويحتوى Crystal Reports على عدد من خيارات التلخيص:

- الحصول على إجمالي القيم فى كل مجموعة.
- الحصول على عدد جميع القيم أو القيم المتميزة.
- معرفة أقصى قيمة، أقل قيمة، متوسط القيم.

لتلخيص بيانات المجموعات:

١. ننقر بزر الماوس الأيمن على مصمم التقرير، نشير إلى Insert ثم ننقر Summary.

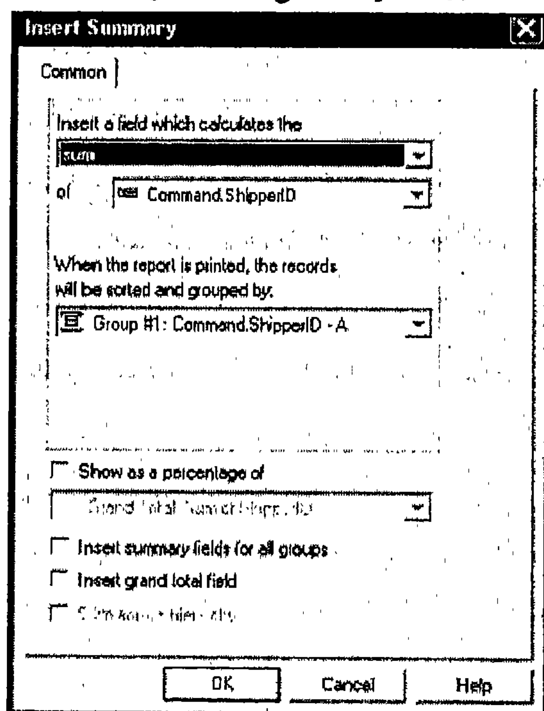
٢. فى مربع حوار Insert Summary المبين بالشكل رقم (٢٩)، نختار عملية التلخيص المستهدفة من القائمة المنسدلة. وتظهر العمليات الحسابية فى هذه القائمة فقط عند اختيار حقل يحتوى على بيانات رقمية فى القائمة المنسدلة الثانية.

٣. فى القائمة المنسدلة الثانية، نختار الحقل الذى يحتوى على القيمة التى نريد تلخيصها.

٤. فى القائمة المنسدلة الثالثة، نختار الحقل الذى نريد الفرز والتجميع على أساسه.

٥. ننقر OK.

على سبيل المثال، عندما نريد الحصول على عدد العملاء فى كل دولة، نكون حقل يعتمد على العميل وتقسم البيانات إلى مجموعات على أساس الدولة.



شكل رقم ٢٩

ترتيب المجموعات على أساس ملخصات القيم

يمكن ترتيب المجموعات على أساس تصاعدي أو على أساس تنازلي باستخدام قيم الملخصات. على سبيل المثال، في تقرير الأوامر، إذا تم تكوين إجماليات فرعية لمبالغ الأوامر على أساس المنطقة، يمكن ترتيب المجموعات:

- من أدنى قيمة إلى قيمة أمر (ترتيب تصاعدي).
- من أعلى قيمة إلى أدنى قيمة (ترتيب تنازلي).

لترتيب المجموعات على أساس قيمة الملخص:

١. ننقر بزر الماوس الأيمن على مصمم التقرير، نشير إلى Report ثم ننقر Top N/Sort Group Expert. ويجب أن يكون لدينا حقل تلخيص في التقرير لكى يصبح خبير Top N متاحا. يظهر مربع حوار Top N Expert محتويا على ملصق لكل واحدة من المجموعات المختصرة في التقرير.

٢. ننقر الملصق الخاص بالمجموعة التى نريد فرزها.

٣. نختار All من القائمة المنسدلة على اليسار.

٤. نختار الملخص الذى نريد بناء اختيارنا علية من القائمة المنسدلة على اليمين.

٥. نحدد اتجاه الفرز بالنقر على Ascending أو Descending.

٦. لاختيار فرز مجموعة ثانية، نكرر الخطوات من ٢ إلى ٥.

عند تشغيل التقرير، سوف يقوم البرنامج بترتيب المجموعات على أساس قيم الملخصات المحددة.

إخفاء التفصيلات فى تقارير الملخصات

عن طريق إخفاء قسم التفصيلات في تقرير ملخص، نتجنب إغراق المستخدمين بالبيانات التى قد لا يحتاجون إليها مباشرة. وعند إخفاء قسم التفصيلات، يقوم المستخدم أولا بتصفح شجرة المجموعات لتحديد مكان البيانات المرغوب فيها. ثم، بالتنقيب فى التقرير، يمكنة الوصول إلى بيانات معينة. لتسهيل عملية التصفح بهذه الطريقة، لا نحتاج

إلا إلى توزيع البيانات على مجموعات ثم إدراج حقول الملخصات فى التقرير.

لإخفاء التفصيلات فى تقرير ملخص :

١. ننقر بزر الماوس الأيمن على قسم Details فى التقرير ثم نختار Format Section.

٢. نختار مربع اختيار Hide (Drill-Down OK).

٣. ننقر OK.

اختيار مجموعة القمة أو مجموعة القاع

قد نحتاج أحيانا إلى إظهار مجموعة القمة أو مجموعة القاع فقط فى التقرير. على سبيل المثال، أسرع خطوط الإنتاج مبيعا، المناطق الأقل مبيعات، وغير ذلك. وبالنظر إلى أن هذا النوع من الاختيار شائع الاستخدام، لذلك يشتمل Crystal Reports على أداة لتنفيذ ذلك بسهولة وسرعة. هذه الأداة هى N/Sort Group Expert .

لاختيار مجموعات القمة أو القاع :

١. ننقر بزر الماوس الأيمن على مصمم التقرير، نشير إلى Report ثم ننقر على Top N/Sort Group Expert. ويجب أن يحتوى التقرير على قيمة ملخص لكى يمكن تنفيذ هذا الاختيار.

٢. ننقر على Top N أو Bottom N من القائمة المنسدلة الأولى.

٣. نختار الملخص الذى نريد بناء اختيارنا علية من القائمة المنسدلة على اليمين.

٤. فى مربع Where N is، ندخل عدد المجموعات التى نريد عرضها. على سبيل المثال :

أ. لتكوين تقرير عن أسرع ثلاثة خطوط إنتاج مبيعا ، نختار Top N فى Top N/Sort Group Expert ونجعل N تساوى ثلاثة.

ب. لتكوين تقرير عن الخمسة مناطق الأقل مبيعا، نختار Bottom N فى Top N/Sort Group Expert ونجعل قيمة N تساوى خمسة.

٥. إذا كنا نريد تكوين مجموعة من جميع السجلات المتبقية، نختار Include Others, With the name وندخل اسم مناسب.

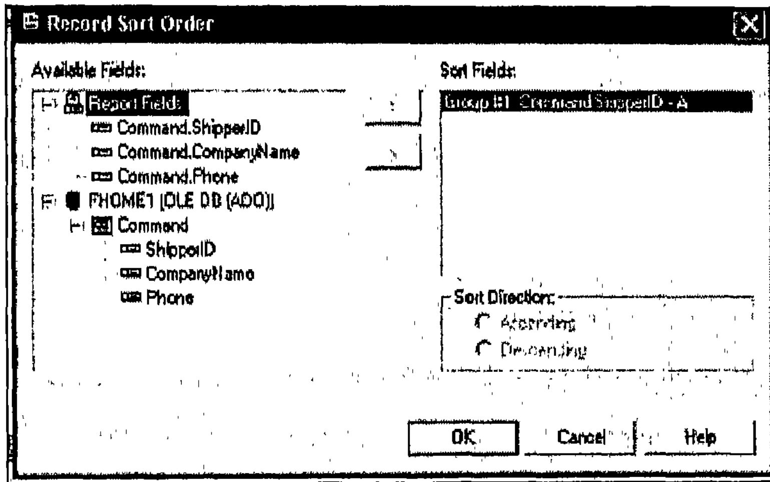
٦. ننقر OK.

عند قيام البرنامج بتشغيل التقرير ، سوف يشتمل فقط على المجموعات التى تم تحديدها.

فرز البيانات

عن طريق فرز السجلات (Sorting)، يمكننا تنظيم البيانات بترتيب معين للمساعدة فى العثور على وتقييم المعلومات. لفرز البيانات، نطبق الخطوات التالية:

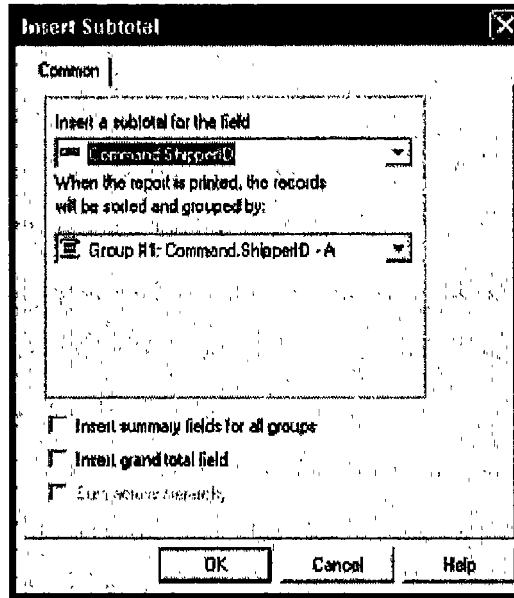
١. ننقر بزر الماوس الأيمن على مصمم التقرير للوصول إلى القائمة المختصرة.
٢. نشير إلى Report وننقر على Sort Records للوصول إلى مربع حوار Record Sort Order المبين بالشكل رقم (٣٠).
٣. فى قائمة Available Fields ، نختار الحقل الذى نريد الفرز على أساسه.
٤. ننقر زر Add. يترتب على ذلك إضافة الحقل إلى قائمة Sort Fields.
٥. فى منطقة Sort Direction بمربع الحوار، ننقر Ascending أو Descinding.
٦. إذا كان الفرز على أساس أكثر من حقل، نختار الحقل الثانى ثم ننقر Add لإضافته إلى حقول الفرز.
٧. مع إضافة كل حقل إلى قائمة Sort Fields، نحدد اتجاه الفرز.
٨. وعند الانتهاء، ننقر زر OK.



شكل رقم ٣٠

تكوين الإجماليات

لمساعدة المستخدمين فى تحليل بيانات تقرير، يمكن تكوي وإضافة حقول إجماليات (Totaling) إلى التقرير. تكوين الإجماليات يمكن أن يكون بسيطا، كما هو الحال عند إعداد إجمالي عام فى نهاية قائمة من السجلات، أو معقدا كما هو الحادث عند عرض إجمالي متحرك بناءا على شروط محددة بواسطة صيغة.



شكل رقم ٣١

لتكوين إجمالي فرعي :

١. ننقر بزر الماوس الأيمن على مصمم التقرير للوصول الى القائمة المختصرة.

٢. نشير إلى Insert ثم ننقر Subtotal.

٣. فـى مـربع حـوار Insert Subtotal المبيـن بالشكل رقم (٣١)، ننقر القائمة المنسدلة ونختار حقل حساب الإجمالي.

٤. فـى القائـمة المنسدلة الثانية، نختار الحقل الذى نريد فرز وتجميع السجلات على أساسه.

٥. إذا ظهرت قائمة منسدلة ثالثة، نحدد اتجاه الفرز.

٦. ننقر OK.

إضافة نسب مئوية إلى تقرير

يمكن حساب النسبة المئوية لإحدى المجموعات بالنسبة لمجموعة أوسع. على سبيل المثال، يمكن عرض المبيعات فى كل مدينة فى صورة نسبة مئوية من إجمالي المبيعات بالنسبة للدولة. ويمكن عرض نسبة مساهمة كل دولة فى إجمالي المبيعات العام.

لحساب نسبة مئوية :

١. ننقر بزر الماوس الأيمن على مصمم التقرير للوصول إلى القائمة المختصرة.

٢. نشير إلى Insert ثم ننقر Summary.

٣. فى مربع حوار Insert Summary، نختار Sum فى القائمة المنسدلة الأولى. ويجب ملاحظة أن دالة Sum لا تظهر إلا إذا كان الحقل المعروض فى القائمة المنسدلة الثانية يحتوى على قيمة رقمية.

٤. فى القائمة الثانية، نختار الحقل الذى نريد حساب الإجمالي له.

٥. فى القائمة الثالثة ، نختار الحقل الذى نستخدمه فى فرز وتجميع السجلات.

٦. إذا ظهرت قائمة رابعة، نحدد فيها إتجاة الفرز.

٧. نختار مربع فحص Show as a percentage of.

٨. فى قائمة Show as a percentage of، نختار المجموعة الذى نريد اتخاذها أساسا لحساب النسبة.

٩. ننقر Ok. يترتب على ذلك إضافة حقل النسبة المئوية إلى التقرير.

تكوين الإجماليات المتحركة

حقول الإجمالي المتحرك، مثل حقول الملخصات، تقدم تحكم أكثر فى كيفية حساب الإجمالي. وتناسب حقول الإجماليات المتحركة الوظائف التالية :

- بيان قيم إجمالي أثناء حسابه مع تتابع السجلات.

- حساب إجمالي لقيمة مستقل عن مجموعات التقرير.
- حساب إجمالي قيمة فى ظل شروط معينة.
- حساب إجمالي قيمة بعد تطبيق صيغة اختيار مجموعة.

وتتم العمليات الحسابية الخاصة بالإجمالي المتحرك على أساس الضوابط المختارة فى Running Total Expert. من ناحية أخرى، يؤثر مكان وضع حقل الإجمالي المتحرك فى القيمة الفعلية التى تظهر على التقرير. على سبيل المثال، عند وضع إجمالي متحرك يقوم بتقييم كل سجل ويستمر بدون إعادة الحساب من البداية، فى قسم مقدمة التقرير (Report Header)، فإن قيمة السجل الأول فقط سوف تظهر. وضع نفس الإجمالي المتحرك فى ذيل التقرير يعيد القيمة الصحيحة. الإجمالي المتحرك يتم حسابه بالطريقة الصحيحة فى كلا الحالتين ، ولكن يتم عرضة حالا فى الحالة الأولى.

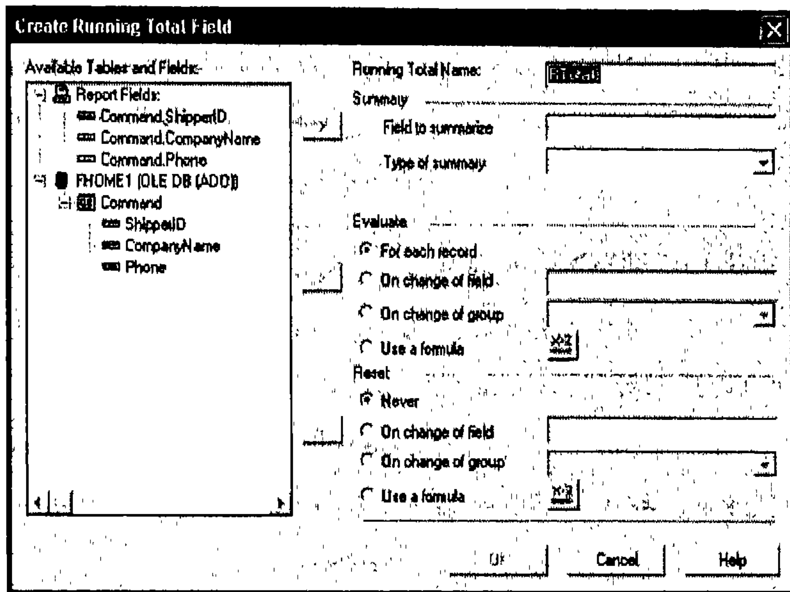
قسم التقرير	التأثير على الإجمالي المتحرك
مقدمة التقرير	يحتوى على السجل الأول فقط.
مقدمة الصفحة	يحتوى على السجلات السابقة والسجل الأول من الصفحة.
مقدمة مجموعة	يحتوى على السجلات السابقة والسجل الأول من المجموعة.
التفصيلات	يكون إجمالي متحرك لكل سجل.
ذيل المجموعة	يكون إجمالي عام لكل مجموعة.
ذيل الصفحة	يحتوى على السجلات السابقة والسجل الأول من الصفحة التالية.
ذيل التقرير	يكون إجمالي عام يشمل جميع سجلات التقرير.

جدول ٢٧

يبين الجدول رقم (٢٧) أقسام التقرير والسجلات التى سوف يستخدمها الإجمالي المتحرك:

تكوين الاجماليات المتحركة فى قائمة

الإجماليات المتحركة هى إجماليات يجرى عرضها عند معالجة كل سجل. وتقوم بتجميع جميع السجلات، سواء فى التقرير، المجموعة، أو غير ذلك. ويشمل الإجمالي المتحرك جميع السجلات حتى السجل الجاري. وأبسط الإجماليات المتحركة هو الإجمالي المتحرك لأحد القوائم. ويستخدم Running Total Expert فى اختيار حقل التلخيص، عملية التلخيص التى نريد تنفيذها، تحديد شروط التقييم، والشرط الذى يتم على أساسه إعادة التقييم من البداية.



شكل رقم ٣٢

لتكوين إجمالي متحرك فى قائمة :

١. فى مربع Field Explorer، ننقر بزر الماوس الأيمن على Running Total Fields، ننقر New. يترتب على ذلك عرض مربع حوار Create Running Total Field المبين فى الشكل رقم (٣٢).

٢. فى مربع حوار Create Running Total Field، ندخل اسم لكائن الإجمالي المتحرك فى حقل Running Total Name.

٣. في منطقة Available Tables and Fields، نختار الحقل الذى نريد تجميعه.

٤. ننقر زر السهم الأول لإضافة الحقل إلى مربع Field to summarize.

٥. في قسم Evaluate بمربع الحوار، نختار وقت تنفيذ الإجمالي المتحرك. بالنسبة للإجمالي المتحرك في قائمة، ننقر On change of field.

٦. في منطقة Available Tables and Fields، نختار حقل "On change of". ننقر السهم الثانى لإضافة الحقل إلى مربع On Change Field.

٧. في قسم Reset بمربع الحوار، ننقر Never. يترتب على ذلك الحصول على إجمالي متحرك لا يعاد تشغيله من البداية في كل مرة. ويعنى ذلك استمرار الإجمالي المتحرك خلال التقرير ويؤدى إلى الحصول على إجمالي عام.

٨. ننقر Ok لحفظ حقل الإجمالي المتحرك. يترتب على ذلك إلى ظهور حقل الإجمالي المتحرك في مربع Field Explorer، تحت Running Total Fields.

٩. نسحب حقل الإجمالي المتحرك إلى قسم Details بالتقرير.

تكوين إجماليات متحركة لمجموعة

يبدأ الإجمالي المتحرك لمجموعة بالسجل الأول في المجموعة و ينتهى بالسجل الأخير. ثم يبدأ مرة أخرى للمجموعة التالية، ثم التالية، حتى نهاية المجموعات. ويجب تقسيم البيانات إلى مجموعات قبل تكوين إجمالي متحرك على أساس المجموعة.

لتكوين إجمالي متحرك على أساس المجموعة :

١. في مربع Field Explorer، ننقر بزر الماوس الأيمن على Running Total Fields ثم New.

٢. في مربع حوار Creating Running Total Field، ندخل الاسم في Running Total Name.

٣. في منطقة Available Tables and Fields، نختار الحقل الذى نريد تجميعه.

٤. ننقر زر السهم الأول لإضافة الحقل إلى مربع Field to summarize.

5. فى قسم Evaluate بمربع الحوار ، ننقر For each record.
6. فـى قسم Reset بمربع الحوار، ننقر On change of group ونقبل الاسم الإفتراضى للمجموعة.
7. ننقر OK لحفظ حقل الإجمالي المتحرك. سوف يترتب على ذلك ظهور الإجمالي المتحرك تحت Running Total Fields بمربع Field Explorer.
8. يمكن سحب حقل الإجمالي المتحرك إلى قسم Details بالتقرير، أو نسحب الحقل المذكور إلى قسم Group Footer، إذا كنا نريد مشاهدة إجمالي عام لكل مجموعة،. ويتم تمييز الإجمالي المتحرك على التقرير بوضع رمز # قبلة.

تكوين إجماليات متحركة شرطية

فى بعض الأوقات، قد يكون لدينا قيم غير مقسمة إلى مجموعات، ونريد فقط إجماليات فرعية لبعض هذه القيم. لتكوين إجمالي متحرك على أساس توفر بعض الشروط:

1. فى مربع Field Explorer، ننقر بزر الماوس الأيمن على Running Total Fields ثم New.
2. فى مربع حوار Creating Running Total Field، ندخل اسم فى Running Total Name.
3. فى منطقة Available Tables and Fields، نختار الحقل الذى نريد تجميعه.
4. ننقر زر السهم الأول لإضافة الحقل إلى مربع Field to summarize.
5. فى قائمة Type of summary، ننقر Sum.
6. فى قسم Evaluate بمربع الحوار، ننقر Use a formula ثم ننقر زر (x+2).
7. فى مربع حوار Running Total Condition Formula ندخل الصيغة فى مربع formula.علىسبيل المثال، لتكوين إجمالي متحرك للمبيعات فى الولايات المتحدة، نستخدم الصيغ التالية.

عند استخدام لغة Crystal:

```
{Customer.Country} = "USA"
```

وعند استخدام لغة Basic:

```
Formula = {Customer.Country} = "USA"
```

١. ننقر زر Save and close.

٢. فى قسم Reset بمربع حوار Create Running Total Field، ننقر Never.

٣. ننقر OK لحفظ حقل الإجمالي المتحرك.

٤. نكون إجمالي متحرك أخر باستخدام الخطوات السابقة. الفرق الوحيد سوف يكون القيمة التى فى الصيغة.

٥. نتعرف على حقول الإجماليات المتحركة فى مربع Field Explorer.

٦. يمكن سحب حقول الإجماليات المتحركة إلى قسم Details بالتقرير، أو إلى قسم Report Footer، عندما نريد مشاهدة إجمالي عام.

استخدام الصيغ

قد نحتاج إلى وضع معلومات على التقرير لا توجد بحقول جداول قاعدة البيانات. فى هذه الحالة نستخدم الصيغ (Formulas) للحصول على البيانات المطلوبة. هناك أربعة مجموعات مختلفة من الصيغ: صيغ التقرير (Report)، صيغ التنسيق المشروط (Conditional Formatting)، صيغ الاختيار (Selection)، وصيغ البحث (Search Formulas). والغالبية من الصيغ الموجودة فى التقرير هى صيغ التقرير (Report Formula)، وصيغ التنسيق المشروط (Conditional Formatting Formula).

صيغ التقرير

صيغ التقرير (Report Formula) هى الصيغ التى نقوم بتكوينها للعمل باستقلالية فى التقرير. على سبيل المثال، الصيغة التى تقوم بحساب عدد الأيام بين تاريخ الأمر وتاريخ الشحن هى صيغة تقرير.

صيغ التنسيق المشروط

تقوم صيغ التنسيق المشروط (Conditional Formatting Formulas) بتغيير خريطة وتصميم التقرير، شكل النص، حقول قواعد البيانات، الكائنات، أو قسم كامل بالتقرير.

وعند الحاجة إلى تكوين صيغة تنسيق، نقوم بالوصول إلى محرر الصيغ (Formula Editor) من خلال محرر التنسيق (Format Editor).

صيغ الاختيار

تحدد صيغ الاختيار (Selection Formulas) السجلات والمجموعات التى تظهر بالتقرير. ومن المعتاد عدم إدخال هذه الصيغ مباشرة، لكن بدلا من ذلك نحدد الاختيار باستخدام Select Expert. يترتب على ذلك قيام Crystal Reports بتوليد صيغ اختيار السجلات والمجموعات. ويمكن تكوين هذه الصيغ يدويا، ولكن يجب استخدام كود Crystal.

صيغ البحث

تساعدنا صيغ البحث (Search Formulas) فى تحديد موقع البيانات فى التقرير. ومثل صيغ الاختيار، من المعتاد عدم إدخال هذه الصيغ مباشرة، وبدلا من ذلك نحدد صيغة البحث باستخدام Search Expert. وبناءا علية، يقوم Crystal Reports بتوليد صيغة البحث المناسبة. وفى حالة الرغبة فى تكوين هذه الصيغ يدويا، يجب استخدام كود Crystal. وإذا كنا على معرفة بكود Basic، فإننا سوف نحتاج إلى معرفة كمية صغيرة من كود Crystal لتعديل معظم صيغ الاختيار والبحث.

إدراج الصيغ

عند تكوين الصيغ، هناك خيار استخدام كود Crystal أو كود Basic. و يمكن تقريبا كتابة أى صيغة باستخدام أحد نوعى الكود إذا كانت الصيغة فى الأصل قد تم كتابتها باستخدام النوع الأخر. ويمكن أن يحتوى التقرير على صيغ تم كتابتها باستخدام كود Crystal وصيغ أخرى تم كتابتها باستخدام كود Basic.

لإدراج صيغة فى تقرير:

١. فى مربع Field Explorer، ننقر بزر الماوس الأيمن على Formula Fields ثم ننقر New.

٢. فى مربع حوار Formula Name، ندخل اسم للصيغة.

٣. ننقر OK.

٤. فى مربع حوار Formula Editor، نختار إما Crystal Syntax أو Basic Syntax.

٥. ندخل الصيغة بطباعتها وباختيار المكونات من أشجار المكونات (Component Trees). ويحتوى محرر الصيغة على شجرة لحقول التقرير (Report Fields)، شجرة للدوال (Functions Tree)، و شجرة العوامل (Operators Tree). ولإضافة أى مكون إلى الصيغة ، نقوم بالنقر المزدوج علية.

٦. ننقر زر Check للتعرف على أى خطأ فى الصيغة.

٧. إذا كانت هناك أخطاء ، نقوم بتصحيحها.

٨. وعندما نتحقق من احتواء الصيغة على الكود الصحيح، ننقر زر Save and close. يترتب على ذلك ظهور الصيغة فى مربع Field Explorer تحت Formula Fields.

٩. نسحب الصيغة ونضعها فى المكان المرغوب به على التقرير. ويجب ملاحظة أن الصيغة الموضوعة على التقرير، يتم تمييز اسمها بوضع الرمز @ قبلها.

أقسام محرر الصيغة

يحتوى محرر الصيغة على عدد من الأجزاء، تشمل مربعات سرد، مربعات سرد مركبة، وقائمة أدوات. تتكون مربعات السرد من أربعة مربعات يبينها الجدول رقم (٢٨).

ويوجد بالقسم الأعلى على اليمين قائمة مركبة (ComboBox) تستخدم فى الاختيار بين كود Crystal وكود Basic للاستخدام فى الصيغة التى نقوم بتكوينها. ويترتب على تغيير الكود المستخدم من Crystal إلى Basic أو العكس إلى تغيير قائمة الدوال فى نافذة Functions ، وبالمثل تغيير العوامل فى نافذة Operators. بينما تبقى حقول التقرير كما هى فى نافذة Report Fields. وتحتوى أشجار Reports Fields، Functions، Operators فى مربع حوار Formula Editor على المكونات الأساسية للصيغ. وعند تكوين الصيغ باستخدام هذه الأداة، يمكننا إضافة أى مكون إلى الصيغة بالنقر المزدوج علية.

المحتويات	النافذة
تشتمل على جميع حقول قاعدة البيانات التى يمكن الوصول إليها بالتقرير. كما تحتوى هذه النافذة على الصيغ والمجموعات السابق تكوينها.	Report Fields
الدوال هى إجراءات سابقة التكوين تعيد قيم وتنفذ العمليات الحسابية مثل المتوسط، الإجمالي، العدد وغيرها.	Functions
تقوم العوامل بوصف عملية أو تصرف يحدث بين اثنين أو أكثر من القيم.	Operators
هى المنطقة التى نستخدمها فى تكوين الصيغة.	Formula text window

جدول ٢٨

تعبيرات SQL

تماثل تعبيرات SQL الصيغ، ولكنها تكتب باستخدام لغة الاستعلام (Structured Query Language). ويمكن استخدام تعبير SQL لاستعلام قاعدة البيانات عن فئة معينة من البيانات. ويمكن أيضا تنفيذ عمليات الفرز ، التجميع، والاختيار باستخدام حقول تعبير SQL.

وتتميز تعبيرات SQL بقدرتها على تحسين أداء التقارير لأن المهمة التى تقوم بها تنفذ عادة على خادم قاعدة البيانات، على عكس الصيغ المعتادة التى يجرى تنفيذها فى الغالب على الكمبيوتر المحلى. ولا يجب استخدام عبارات SQL بكثافة، لأن Crystal Reports

تحتوى على لغة الصيغ الخاصة بها والتى هى فى نفس الوقت أكثر قوة من SQL المعتادة. ولكن فى بعض الحالات، يكون من الضروري استخدام حقول تعبيرات SQL لزيادة سرعة عمليات معالجة التقارير.

تكوين حقل تعبير SQL:

١. فى مربع Field Explorer، ننقر بزر الماوس الأيمن على SQL Expression Fields ثم New. ويجب ملاحظة أن بند SQL Expression Fields لا يظهر بمربع Field Explorer إلا فى حالة استخدام مصدر بيانات SQL فى تصميم التقرير.

٢. ندخل اسم فى مربع Name ثم ننقر OK. يترتب على ذلك ظهور SQL Expression Editor.

٣. نطبع التعبير الذى نريد إدخاله فى SQL Expression Editor.

٤. ننقر OK.

تنسيق البيانات

يشير التنسيق (Formatting) إلى التغييرات فى خريطة تصميم التقرير، وبالمثل شكل ظهور النصوص، الكائنات، أو أقسام التقرير بالكامل. ويمكن استخدام التنسيق للقيام بالكثير من الأعمال التى من بينها :

- توجيه الإنتباة إلى بيانات معينة.
- إضفاء مظهر احترافي على التقرير.
- تغيير عرض التواريخ، الأرقام، القيم المنطقية، قيم العملة، وسلاسل النصوص.

وللقيام بتنسيق كائن أو قسم فى تقرير، ننقر بزر الماوس الأيمن علية ثم نختار Format. ولتنسيق كائن رسم بياني، ننقر بزر الماوس الأيمن علية ثم نختار Chart Expert. ويمكن أيضا استخدام نافذة Properties فى Visual Studio .NET لتنسيق الكائنات والأقسام.

تنسيق الأرقام والتواريخ

لتدعيم المفاهيم المستخدمة فى مهنة المحاسبة، يسمح لنا Crystal Reports بتقرير

كيفية عرض رموز العملة، القيم السالبة، والأصفار على التقارير المالية. ويمكن أيضا ضبط التقرير لعكس الإشارات الخاصة بالمبالغ المدينة والدائنة.

لتنسيق الأرقام والتواريخ:

١. ننقر بزر الماوس الأيمن على حقل العملة، الرقم، أو التاريخ ثم نختار Format.

٢. نختار واحدا من خيارات التنسيق، أو ننقر Customize لتحديد تنسيق أخر.

إضافة حدود ، ألوان ، أو ظلال إلى الحقول

يسمح لنا Crystal Reports بإضافة حدود، لون، أو ظلال إلى حقول التقرير لكى يتم التركيز على البيانات المهمة وتكوين تقارير محترفة.

لإضافة حدود ، ألوان ، وظلال:

١. ننقر بزر الماوس الأيمن على الكائن ونختار Format.

٢. ننقر ملصق Border ونختار نمط الخط، اللون، ولون الخلفية للحقل.

٣. ننقر OK لحفظ التغييرات.

التنسيق المشروط

يطبق هذا النوع من التنسيق فى ظل توفر شروط معينة. على سبيل المثال، فى أحد التقارير يمكن أن نحتاج إلى:

- طباعة أرصدة العملاء باللون الأحمر عند تجاوز تاريخ استحقاق السداد.
- طباعة التاريخ بصيغة تناسب المستخدم للتقرير.
- إظهار لون الخلفية كل ثلاثة سطور.

يمكننا تحديد التنسيق الشرطي باستخدام الصيغ. وعند تكوين صيغة تنسيق شرطية ، يكون لها الأسبقية على جميع قيم الضبط التى تم إعدادها فى مربع حوار Format Editor .

لإضافة تنسيق شرطي إلى كائن:

١. ننقر بزر الماوس الأيمن على الكائن، نختار Format.

٢. ننقر زر الصيغة المناسب الموجود فى الجانب الأيمن بمربع الحوار.

٣. فى مربع حوار Format Editor، ندخل الصيغة.

٤. ننقر Save. إذا كان هناك أخطاء، سوف يقوم Crystal Reports بعرض رسالة بهذا الخطأ وطلب الموافقة على الخروج أو فحص الخطأ.

٥. ننقر OK للعودة التى التقرير.

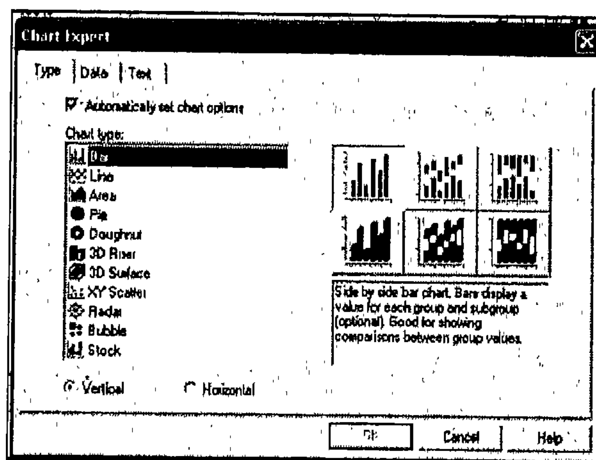
ترقية عرض التقرير

يشرح هذا القسم الوسائل التي يمكن استخدامها لجذب الإنتباة إلى البيانات؛ تغيير عرض التواريخ، الأرقام، والقيم الأخرى؛ إخفاء الأقسام غير المرغوب بها؛ والقيام بمهام تنسيق أخرى تضفى على التقرير مظهر تقارير المحترفين.

إدراج الرسوم البيانية

يتيح Crystal Reports إمكانية وضع رسوم بيانية معقدة وملونة على التقرير. ولا تعتبر الرسوم البيانية وسيلة لعرض البيانات فقط، بل إنها أداة تحليلية أيضا. ويمكن للمستخدمين التنقيب باستخدام الرسم البياني ، أو مفتاح الرسم البياني، للحصول على معلومات تفصيلية. ويمكن إعداد الرسوم البيانية باستخدام ما يلى:

- حقول الملخصات والإجماليات الفرعية.
- حقول التفصيلات، الصيغ، والإجماليات المتحركة.
- الملخصات المتقاطعة.



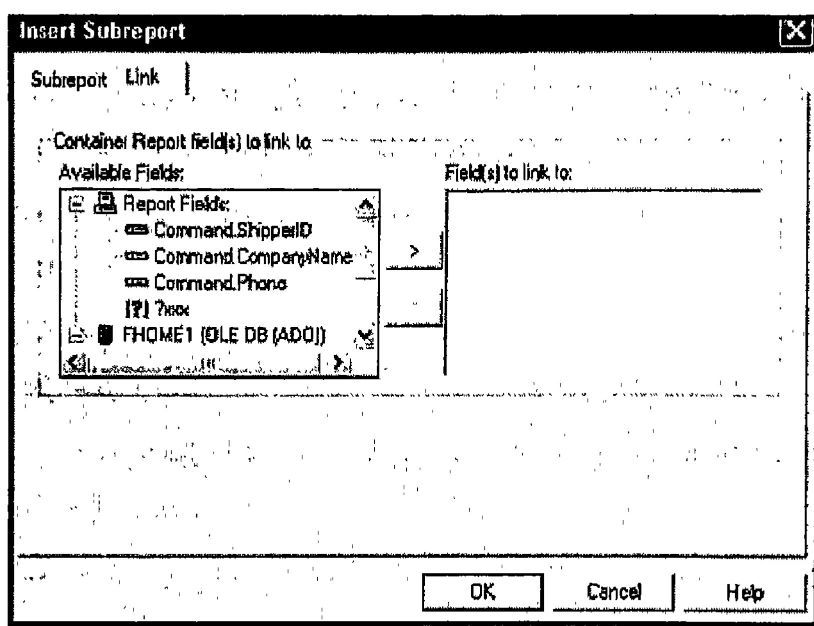
شكل رقم ٣٣

لإدراج رسم بياني:

١. ننقر بزر الماوس الأيمن على مصمم التقرير، نشير إلى Insert ثم ننقر Chart. يترتب على ذلك عرض مربع حوار Chart Expert المبين بالشكل رقم (٣٣).
٢. فى ملصق Type بمربع حوار Chart Expert، نختار نوع الرسم.
٣. نختار Automatically set chart options عندما نريد استخدام الضوابط الافتراضية.
٤. ننقر ملصق Data.
٥. فى منطقة Placement، نحدد كيفية ظهور الرسم البياني على التقرير، ثم ننقر Header أو Footer لتحديد مكان وضع الرسم.
٦. فى منطقة Layout، نختار Chart Layout.
٧. فى منطقة Data، نحدد حقول قاعدة البيانات المستخدمة شروطا.
٨. إذا ظهرت ملصقات Axes و Options، يمكننا تعديل بعض خصائص الرسم، مثل التصاعد، المفتاح، ونقاط البيانات.
٩. ننقر ملصق Text، ونقبل معلومات العنوان الافتراضية أو نضيف عناوين جديدة إلى الرسم.
١٠. ننقر OK.

إدراج التقارير الفرعية

التقرير الفرعى (Subreport) هو تقرير داخل تقرير. وعن طريق استخدام التقارير الفرعية، يمكن إدماج التقارير التى لا يوجد بينها علاقات فى تقرير واحد. ويمكن بهذه الطريقة تنسيق البيانات التى لايمكن تنسيقها بطريقة أخرى، أو يمكن عرض مشاهد مختلفة من نفس البيانات فى تقرير واحد.



شكل رقم ٣٤

وإذا كان التقرير به قسم يقوم بمعالجة عدد ضخم من السجلات، يمكن وضع هذا القسم فى تقرير فرعى تحت الطلب. التقرير الفرعى الموضوع تحت الطلب يظهر فى صورة ارتباط نشط (hyperlink) فى التقرير الأساسى. وعند فتح التقرير الأساسى، لا يتم استخراج بيانات بالنسبة للتقرير الفرعى إلى أن يتم النقر على الارتباط النشط.

لإدراج تقرير فرعى:

١. ننقر بزر الماوس الأيمن على مصمم التقرير، نشير إلى Insert، ثم ننقر Subreport.

٢. نسحب كائن التقرير إلى التقرير.

٣. نختار تقرير أخر فى المشروع، أو نكون تقرير جديد خاص بالتقرير الفرعى.

٤. نختار On-demand Subreport لكى يمكننا استخراج البيانات الخاصة بالتقرير الفرعى عند الحاجة. وإلا، فإن كل بيانات التقارير الفرعية سوف تظهر مع التقرير. ويجب ملاحظة أن استخدام التقارير الفرعية عند الطلب سوف يؤدى إلى زيادة أداء التقارير التى تحتوى على تقارير فرعية.

٥. ننقر ملصق Link، إذا كنا نريد ربط التقرير الفرعى مع البيانات فى التقرير الأساسي. يترتب على ذلك عرض صفحة Container Report Fields to Link بمربع حوار Insert Subreport المبينة بالشكل رقم (٣٤).

٦. نختار الحقل الذى نريد إستخدامة حقل ارتباط فى التقرير الأساسى من قائمة Available fields.

٧. نستخدم قسم Field link، الذى يظهر فقط عند اختيار حقل ارتباط، لضبط الارتباط لكل حقل ارتباط:

أ. نختار الحقل الذى نريد ربطة مع التقرير الأساسى من Subreport parameter field to use.

ب. نختار مربع فحص Select data based on field ونختار حقل من القائمة المنسدلة المجاورة لترتيب بيانات التقرير الفرعى بناءا على حقل محدد. وإذا لم يتم تحديد أى شئ هنا، فإن التقرير الفرعى سوف يستخدم تنظيم التقرير الأساسى.

٨. ننقر OK.

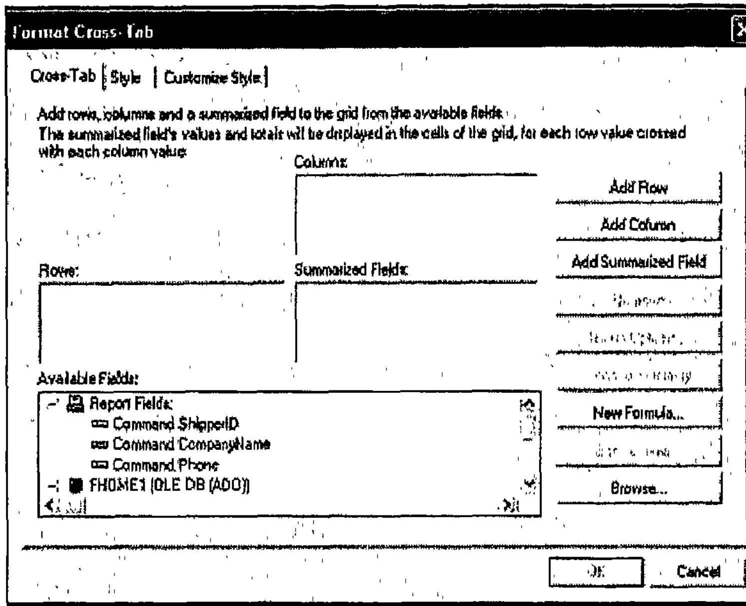
عند تشغيل التقرير، سوف يقوم البرنامج بتنسيق بيانات التقرير الأساسى مع بيانات التقرير الفرعى.

التقارير الفرعية غير المرتبطة

التقارير الفرعية غير المرتبطة (Unlinked Reports) هى تقارير مستقلة، لا يتم تنسيق بياناتها بأي طريقة مع بيانات التقرير الأساسى. وفى التقارير غير المرتبطة، لا توجد أي محاولات لموافقة السجلات فى تقرير مع السجلات فى التقرير الأخر. وليس من الضروري أن تقوم التقارير الفرعية غير المرتبطة باستخدام نفس البيانات مثل التقرير الأساسى، يمكنها استخدام نفس مصدر البيانات أو مصدر بيانات مختلف بالكامل. بالإضافة إلى ذلك، فإن التقرير الفرعى ليس ملزما باستخدام جدول واحد. التقرير الفرعى غير المرتبط يمكن أن يستخدم جدول واحد أو عدة جداول.

التقارير الفرعية المرتبطة

التقارير الفرعية المرتبطة (Linked Reports) تمثل الجانب الأخر؛ حيث يتم تنسيق بياناتها مع بيانات التقرير الأساسى. يقوم البرنامج بموافقة السجلات فى التقرير الفرعى مع السجلات فى التقرير الأساسى. إذا قمنا بتكوين تقرير أساسى يحتوى على معلومات العملاء وتقرير فرعى يحتوى على معلومات أوامر المبيعات ثم جرى الربط بينهما، يقوم البرنامج بتكوين تقرير فرعى لكل عميل يحتوى على كل الأوامر الخاصة بالعميل.



شكل رقم ٣٥

إدراج كائنات المعلومات المتقاطعة

كائن المعلومات المتقاطعة (Cross-Tab) هو عبارة عن شبكة تعيد معلومات على أساس معيار يجرى تحديده. ويتم عرض البيانات فى أعمدة وصفوف. هذا التنسيق يمكن المستخدمين من مقارنة البيانات والتعرف على الاتجاهات. ويتكون كائن المعلومات المتقاطعة من ثلاثة عناصر : الصفوف، الأعمدة، وحقول الملخصات.

لإدراج كائن معلومات متقاطعة :

١. ننقر بزر الماوس الأيمن على مصمم التقرير، نشير إلى Insert ثم ننقر Cross-Tab.

٢. نضع كائن Cross-Tab على التقرير.

٣. فـى مـربع حـوار Format Cross-Tab المـبين بالشكل رقم (٣٥)، نضيف حقول إلى مناطق الصفوف، الأعمدة، وحقول الملخصات.

٤. نـنقر ملصـق Style لاختـيار التصـميم المناسب، أو ملصق Customize Style لتكوين تصميم خاص بنا.

إخفاء أقسام التقرير

يحتوى Crystal Reports على ثلاثة خصائص يمكن استخدامها لإخفاء أقسام التقرير.

خاصية Hide

تقوم هذه الخاصية بإخفاء قسم عند تشغيل التقرير. على سبيل المثال، فى تقرير ملخص، يمكن استخدام هذه الخاصية لعرض الملخصات فقط، وليس التفصيلات خلف هذه التفصيلات. وعند تطبيق هذه الخاصية على قسم، فان التفصيلات بهذا القسم تصبح مرئية عند استخدام مؤشر Drill-down للتنقيب عن محتويات القسم. وهذه الخاصية مطلقة ولا يمكن ربط استخدامها بتحقق أحد الشروط عن طريق الصيغ. لاستخدام هذه الخاصية:

١. ننقر بزر الماوس الأيمن على التقرير.

٢. من القائمة المختصرة، نختار Format Section.

٣. فى مربع حوار Section Expert، نختار Hide (Drill-Down-OK).

خاصية Suppress

تقوم هذه الخاصية أيضا بإخفاء قسم عند تشغيل التقرير. وعلى خلاف خاصية Hide، لا تمكننا هذه الخاصية من التنقيب عن التفصيلات. ويمكن تطبيق هذه الخاصية بصفة مطلقة أو عند توفر أحد الشروط باستخدام الصيغ. لاستخدام هذه الخاصية:

١. ننقر بزر الماوس الأيمن على التقرير.

٢. من القائمة المختصرة، نختار Format Section.

٣. ننقر بزر الماوس الأيمن على قسم التقرير ثم نختار خاصية Suppress(No Drill-Down).

•

خاصية Suppress Blank Section

تقوم هذه الخاصية بإخفاء أحد الأقسام عندما لا يحتوى على شئ بداخلة. وإذا تم وضع شئ ما داخل القسم، فإن القسم سوف يصبح مرئيا. لاستخدام هذه الخاصية:

١. ننقر بزر الماوس الأيمن على التقرير.

٢. من القائمة المختصرة، نختار Format Section.

٣. فى مربع حوار Section Expert، نختار خاصية Suppress Blank Section.

التحكم فى التقارير وقت التشغيل

نحتاج وقت تشغيل التطبيقات وعرض التقارير إلى الاستجابة لمتطلبات المستخدمين التى قد تشمل اختيار التقرير المطلوب تشغيله، تغيير شكل أداة عرض التقارير، تغيير شكل التقرير ونوع البيانات المعروضة. لتمكين المستخدم من تنفيذ هذه المهام، يوفر Crystal Reports إمكانية استخدام الكائنات (Objects) للقيام بمختلف مهام تكوين التقارير. كما يوفر لغة صياغة قوية يمكن استخدامها لتغيير طريقة عرض التقرير والتحكم فى البيانات التى يحتوى عليها. يناقش هذا القسم عملية التفاعل مع أحداث المستخدمين والوسائل المتوفرة للقيام بذلك.

استخدام الكود

يحتوى Crystal Reports على العديد من التصنيفات (Classes) المستخدمة فى تكوين الكائنات المختلفة المستخدمة فى تنفيذ الوظائف التى يقوم بها. كما يحتوى على لغة صياغة قوية تستخدم فى تكوين الصيغ المختلفة. وعن طريق خصائص (Properties) الكائنات، الوسائل (Methods) التى تحتوى عليها، والأحداث (Events) التى تستجيب لها، يمكن الوصول إلى التقارير وتغيير خصائصها فى وقت التشغيل.

استخدام الكائنات

هناك الكثير من الكائنات التى يحتوى عليها Crystal Reports، غير أن أهم هذه الكائنات بالنسبة للمبرمج هى كائن ReportDocument، وكائن CrystalReportViewer.

كائـن *ReportDocument*

يقدم كائن ReportDocument أدوات التحكم الأساسية فى التقارير. ويحتوى على الكثير من الخصائص والوسائل التى تسمح لنا بالتحكم فى اختيار التقرير وشكل عرضة من خلال ضبط الخصائص والاستجابة للأحداث. ويمكن استخدام هذا الكائن فى إجراء التعديلات المختلفة على التقرير باستخدام الكود ثم تمرير التقرير إلى كائن أخر هو كائن CrystalReportViewer لعرضة أمام المستخدمين. لفتح ملف تقرير، يمكن استخدام وسيلة Load أو تخصيص تقرير نوعى لهذا الكائن. ولكى نستخدم هذا الكائن، يجب تكوين مرجع إلى منطقة أسماء CrystalDecisions.CrystalReports.Engine. ويتم إضافة هذا المرجع تلقائيا عند إضافة تقرير إلى التطبيق الذى نعمل به.

خصائص ReportDocument

يحتوى كائن ReportDocument على الكثير من الخصائص التى تسمح لنا بالتحكم فى شكل وسلوك التقارير. فيما يلى أهم الكائنات والمجموعات التى يحتوى عليها:

- كائن Database. تتيح الوصول إلى معلومات قاعدة البيانات التى يحتوى عليها التقرير. ويحتوى كائن Database على مجموعة Tables. تحتوى هذه المجموعة على كل كائنات الجداول المستخدمة فى التقرير. ويوفر لنا كائن Table إمكانية ضبط معلومات الاتصال والحصول عليها من خلال كائن ConnectionInfo. كما يوفر المعلومات عن كل الحقول المتاحة فى الجدول من خلال مجموعة DatabaseFieldDefinitions.
- كائن DataDefinition. يتيح هذا الكائن الوصول إلى الحقول، حقول المعاملات، حقول الفرز، حقول أسماء المجموعات، حقول الملخصات، حقول الإجماليات المتحركة، وحقول تعبيرات SQL. ويمكن عن طريق هذا الكائن الوصول إلى

المجموعات التى يحتوى عليها التقرير.

- كائن ExportOptions. يوفر هذا الكائن الخصائص المتعلقة باستخراج وضبط خيارات تصدير التقرير. ويستخدم هذا الكائن لضبط خيارات جهة التصدير، ونوع التصدير.
- كائن PrintOptions. يحتوى هذا الكائن على الخصائص والوسائل المستخدمة فى ضبط خيارات طباعة التقرير. من الخصائص التى يمكن ضبطها، خاصية اسم الطابعة، حجم الورقة، وهوامش الورقة.
- كائن ReportDefinition. يسمح هذا الكائن باستخراج كل المناطق، كائنات التقرير، والأقسام فى التقرير. يتيح لنا ذلك إمكانية استخراج وضبط خيارات الصياغة لتلك البنود.
- كائن ReportOptions. يسمح لنا هذا الكائن بالحصول على البيانات المتعلقة بخيارات خاصة بالتقرير.

ويحتوى كائن ReportDocument على حدث وحيد هو ReportDocument.InitReport. يقع هذا الحدث عند نجاح تحميل التقرير، ويمكن تكوين إجراء معالجة لهذا الحدث. المثال التالى يوضح كيفية ربط الحدث مع كائن ReportDocument، وكيفية ضبط نص متحكم للعرض عندما يتم تحميل التقرير:

```
Dim WithEvents report As New ReportDocument ()
Private Sub report_InitReport _
(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles report.InitReport
  Label1.Text = " Report has been loaded."
End Sub
```

كائن CrystalReportViewer

يحتوى هذا الكائن على الخصائص التى تمكننا من التحكم فى شكل وسلوك التقرير. وأهم الكائنات التى يحتوى عليها هذا الكائن، تشمل الكائنات التالية:

- كائن TableLogOnInfos. يوفر الوصول إلى مجموعة TableLogOnInfo. وهذا

الكائن بدورة يوفر الخصائص اللازمة للحصول على وضبط خيارات الاتصال مع الجدول، مثل اسم الخادم، اسم المستخدم، اسم قاعدة البيانات، وكلمة المرور.

- كائن ParameterFields. يوفر الوصول إلى مجموعة ParameterField. وتسمح لنا هذه المجموعة بالحصول على معلومات المعاملات فى التقرير وضبطها. على سبيل المثال، يمكن ضبط خصائص للحصول على خيارات وقيم حقل معاملات وضبطها، مثل القيم الحالية، القيم الافتراضية، والنص الحث المستخدم.
- خاصية ReportSource. تستخدم هذه الخاصية لضبط مصدر التقرير. ويمكن أن يكون هذا المصدر، كائن ReportDocument، أو سلسلة تحتوى على موقع الملف الخاص بالتقرير.

ويحتوى هذا الكائن على عدد من الأحداث، طبقا للتفصيل التالى:

- حدث CrystalReportViewer.Drill. يقع هذا الحدث عند التنقيب عن المعلومات بالتقرير. الكود التالى، يوضح كيفية ربط اسم المجموعة المستخدمة فى التنقيب مع نص متحكم Label:

```
Private Sub CrystalReportViewer1_Drill (ByVal source As Object, ByVal e As _
CrystalDecisions.Windows.Forms.DrillEventArgs) Handles _
CrystalReportViewer1.Drill
  Label1.Text = e.CurrentGroupName
End Sub
```

- حدث CrystalReportViewer.DrillDownSubreport. يقع هذا الحدث عند التنقيب فى تقرير فرعي. الكود التالى يوضح كيفية ضبط متحكم Label على اسم التقرير الفرعى:

```
Private Sub CrystalReportViewer1_DrillDownSubreport (ByVal source As Object, _
ByVal e As CrystalDecisions.Windows.Forms.DrillSubreportEventArgs) Handles _
CrystalReportViewer1.DrillDownSubreport
  Label1.Text = e.CurrentSubreportName
End Sub
```

- حدث CrystalReportViewer.HandleExecption. يقع عند حدوث خطأ. الكود التالى يعرض رسالة الخطأ عند حدوثه.

```
Private Sub CrystalReportViewer1_HandleException (ByVal source As _ Object, ByVal e As CrystalDecisions.Windows.Forms.ExceptionEventArgs) _ Handles CrystalReportViewer1.HandleException
  Label1.Text = e.Exception.ToString;
End Sub
```

- حدث CrystalReportViewer.Navigate. يقع هذا الحدث عندما يقوم المستخدم بالتجول فى التقارير. الكود التالى يضبط متحكم Label على رقم الصفحة عند التجول فى التقرير:

```
Private Sub CrystalReportViewer1_Navigate (ByVal source As Object, _ ByVal e As CrystalDecisions.Windows.Forms.NavigateEventArgs) Handles _ CrystalReportViewer1.Navigate
  Label1.Text = e.NewPageNumber
End Sub
```

- حدث CrystalReportViewer.ReportRefresh. يقع هذا الحدث عند تجديد معلومات التقرير. الكود التالى يعرض رسالة عند تجديد بيانات التقرير:

```
Private Sub CrystalReportViewer1_Refresh (ByVal source As Object, _ ByVal e As CrystalDecisions.Windows.Forms.ViewerEventArgs) Handles _
CrystalReportViewer1.Refresh
  Label1.Text = "Data is refreshed."
End Sub
```

- حدث CrystalReportViewer.Search. يقع هذا الحدث عند البحث عن نص فى التقرير. الكود التالى يعرض رسالة عن النص المستخدم فى البحث.

```
Private Sub CrystalReportViewer1_Search (ByVal source As Object, _
 ByVal e As CrystalDecisions.Windows.Forms.SearchEventArgs) Handles _
CrystalReportViewer1.Search
  Label1.Text = e.TextToSearch
End Sub
```

- حدث CrystalReportViewer.ViewZoom. يقع هذا الحدث عند تغير معامل التكبير والتصغير.

لغة الصياغة

يمكن استخدام لغة Crystal Reports أو لغة Basic فى تكوين الصيغ (Formulas). ويمكن كتابة الصيغ المكتوبة بلغة Crystal Reports، باستخدام لغة Basic أو العكس. كما أن التقارير يمكن أن تحتوى على صيغ مكتوبة باستخدام لغة Crystal Reports وأخرى مكتوبة باستخدام لغة Basic. غير أن الصيغة الواحدة يجب أن تستخدم لغة واحدة. وتماثل لغة Basic المستخدمة فى Crystal Reports لغة Visual Basic، غير أنها تحتوى على بعض الإضافات الخاصة بالتقارير. وحيث أننا بصدد الحديث عن Visual Basic، لذلك سوف نقتصر على إيضاح لغة Basic المستخدمة فى Crystal Reports.

استخدام كود *Basic*

يمكن أن تعمل التقارير التى تستخدم كود Basic فى تكوين الصيغ على أى جهاز كمبيوتر يعمل علية Crystal Reports. ويتميز استخدام كود Basic بدلا من كود Crystal Reports فى إعداد صيغ Crystal Reports، بأنة لا يتطلب توزيع ملفات إضافية مع التطبيقات التى تستخدم التقارير. وتتوفر السمات التالية فى لغة Basic المستخدمة لإعداد صيغ Crystal Reports:

- تستخدم تقريبا نفس الدوال المستخدمة فى Visual Basic. يشمل ذلك دوال السلاسل، مثل دالة Len، دالة Mid، وغيرها. كما تشمل الدوال الحسابية، الدوال المالية، دوال التاريخ.
- تدعم معظم العوامل المستخدمة فى Visual Basic، مثل عامل وصل السلاسل (&) وصيغ التاريخ.
- تستخدم معظم العبارات وهياكل التحكم المستخدمة فى Visual Basic، مثل Select، Do Until، Do While، Do Until، While، For/Next.
- تستخدم نفس كود الملاحظات ووصل سطور الكود المستخدمة فى Visual Basic.

نتيجة الصيغة

نتيجة الصيغة (Formula) هى القيمة التى يتم طباعتها عند وضع الصيغة التقرير،

وتسمى القيمة العائدة من الصيغة. ويجب أن تعيد أى صيغة فى Crystal Reports قيمة معينة. يقوم كود Basic بتنفيذ ذلك عن طريق ضبط قيمة متغير خاص يسمى Formula. على سبيل المثال، الكود التالى يمثل صيغة Basic بسيطة:

```
Formula = 10
```

ويمكن أن تكون القيمة العائدة من الصيغة من أحد أنواع سبعة: Number، Currency، String، Boolean، Date، Time، Date and Time. ويجب تخصيص قيمة للمتغير Formula لكى تكون الصيغة صحيحة. من أمثلة الصيغ والقيمة العائدة منها، ما يلى:

```
Global x As String, y As Number, z As DateTime
x = "hello"
y = 10.5
z = #Aug 6, 1976#
formula = 10
```

ويمكن ضبط متغير Formula عدة مرات داخل الصيغة الواحدة، ولكن لايجب تغيير نوع بيانات القيمة التى تم تخصيصها له فى المرة الأولى. وكود Basic المستخدم فى إعداد الصيغ غير حساس بالنسبة لأحجام الحروف، ولهذا تتساوى الكلمات المكتوبة بنفس الحروف سواء كانت الحروف صغيرة أو كبيرة.

الحقول

هناك الكثير من الحقول (Fields) التى نستخدمها عند تكوين التقرير، يمكن استخدامها فى الصيغ التى نقوم ببنائها. على سبيل المثال، يمكن استخدام حقول قاعدة البيانات، حقول المعاملات، حقول الإجمالي المتحرك، حقول عبارات SQL، حقول الملخصات، وحقول المجموعات يمكن استخدامها فى الصيغ. ويمكن استخدام حقول الصيغ الأخرى. وتظهر أسماء الحقول داخل الصيغة محاطة بأقواس{}. وتظهر حقول قاعدة البيانات مؤهلة باسم الجدول الذى تتبعة. وتوضع العلامات التالية فى مقدمة الحقول: علامة ? فى حقول المعاملات، علامة @ فى حقول الصيغ الأخرى، علامة # فى حقول الإجمالي المتحرك، علامة ٪ فى حقول تعبيرات SQL. وتظهر حقول الملخصات وحقول المجموعات مثل كود استدعاء الدوال، كما يتضح من الكود التالى:

```
Sum ({Orders.Order Amount}, {Orders.Ship Via})
```

```
GroupName ({Orders.Ship Via})
```

الصيغة التالية تقوم بحساب عدد الأيام منذ تاريخ إعداد طلب الشراء إلى تاريخ شحن البضاعة :

```
Rem A formula that uses database fields
formula = {Orders.Ship Date} - {Orders.Order Date}
```

وتقوم الصيغة التالية بحساب إجمالي قيمة البضاعة المطلوبة :

```
formula = {Orders Detail.Unit Price} * _
        {Orders Detail.Quantity}
```

وتقوم الصيغة التالية بحساب سعر البيع على أساس أنة ٨٠٪ من السعر الأصلى

```
formula = {Orders Detail.Unit Price} * 0.80
```

العبارات

يتكون الكود الذى تتكون منة الصيغ من سلسلة من العبارات (Statements). ويجب فصل كل عبارة عن العبارة السابقة بسطر جديد أو باستخدام رمز الوقف الإستدراكى (:). ومن المعتاد وضع كل عبارة على سطر منفصل، ويكن استمرار العبارة على السطر التالى باستخدام شرطة أسفل السطر. فيما يلى أمثلة من العبارات:

```
Dim x As Number
x = 10 + 10 + 10
x = 10 + _
  10 + 10
y = "Hello" : x = 30 : formula = True
```

المتغيرات

عندما تستخدم الصيغة أحد المتغيرات (Variables)، فإنها تبحث عن قيمة المتغير وتستخدمها. وتحتفظ المتغيرات بالقيم التى يتم تخصيصها لها إلى أن يتم تخصيص قيم جديدة لها. ويجب الإعلان عن المتغيرات قبل استعمالها لكى يصبح Crystal Reports على علم بها. الأمثلة التالية توضح استخدام المتغيرات فى الصيغ.

الكود التالى يقوم باستخراج رقم المنطقة من رقم الفاكس الخاص بأحد العملاء:

```
Dim areaCode As String
areaCode = Left ({Customer.Fax}, 3)
```

نطاق القيم

يمكن تكوين نطاق (Range) من كل أنواع القيم البسيطة فيما عدا القيم المنطقية. الأمثلة التالية توضح تكوين واستخدام نطاق من القيم. المثال الأول يحتوى على نطاق يتكون من ٢ إلى ٥ وما بينهما من أرقام:

```
2 To 5
```

والنطاق التالى يتكون من ٢ إلى ٥ ولا يشمل الرقم ٢:

```
2 _To 5
```

والنطاق التالى يتكون من كل الأرقام التى أقل من أو تساوى ٥:

```
Is <= 5
```

والنطاق التالى يتكون من عدة تواريخ:

```
#Jan 5, 1999# To #Dec 12, 2000#
Is >= #Jan 1, 2000#
```

ويستخدم نطاق القيم فى الصيغ غالبا مع عبارات If و Select. المثال التالى يقوم بحساب درجات شهادة الطالب بناءا على درجات الامتحانات الخاصة به. حيث يتم تخصيص "A " للنتيجة التى تساوى ٩٠ أو أكبر منها. وتخصيص "B" للدرجات من ٨٠ إلى ٩٠ بدون أن تشمل الدرجة ٩٠:

```
Select Case {Student.Test Scores}
  Case Is >= 90
    formula = "A"
  Case 80 To_ 90
    formula = "B"
  Case 70 To_ 80
    formula = "C"
  Case 60 To_ 70
    formula = "D"
  Case Else
    formula = "F"
End Select
```

ويختبر المثال التالى وجود قيمة معينة فى نطاق باستخدام عامل In:

```
formula = 5 In 2 To 10
```

```
formula = 5 In 2 To_ 5
formula = 5 In 2 To 5
```

وللحصول على نهايات النطاق، نستخدم دالة Maximum أو دالة Minimum:

```
formula = Maximum (2 To 10)
```

المصفوفات

المصفوفات (Arrays) هى قوائم تحتوى على قيم من نفس النوع. ويطلق على هذه القيم عناصر المصفوفة. ويمكن أن تكون قيم العناصر من أنواع البيانات البسيطة أو من نوع نطاق بيانات (Range). ونقوم بتكوين المصفوفة عن طريق استخدام دالة Array. وتكون المصفوفات أكثر فائدة عندما تحتوى على متغيرات بسبب توفر إمكانية تغيير قيم العناصر فى تلك المصفوفات. فيما يلى أمثلة توضح استخدام المصفوفات فى الصيغ.

الكود التالى يوضح تكوين مصفوفة من ثلاثة قيم ثابتة:

```
Array (10, 5, 20)
```

الكود التالى يوضح تكوين مصفوفة من سبعة قيم من نوع السلاسل:

```
Array ("Sun", "Mon", "Tue", "Wed", "Th", "Fri", "Sat")
```

الكود التالى يبين تكوين مصفوفة من نطاقين من التواريخ:

```
Array (#Jan 1, 1998# To #Jan 31, 1998#, _
    #Feb 1, 1999# To #Feb 28, 1999#)
```

ويمكن استخراج قيم العناصر من المصفوفة باستخدام الأقواس التى تحتوى على فهرس العنصر المطلوب. الكود التالى يستخرج القيمة 5 من مصفوفة:

```
Array (10, 5, 20) (2)
```

ويمكن أيضا استخدام نطاق قيم لاستخراج عناصر مصفوفة. الكود التالى يستخدم نطاق قيم لاستخراج المصفوفة (5,20) من المصفوفة (10,5,20):

```
Array (10, 5, 20) (2 To 3)
```

هياكل التحكم فى تنفيذ الكود

عندما لا تحتوى الصيغة على هياكل تحكم، يتم تنفيذ التعليمات بداخلها بالترتيب من العبارة الأولى إلى العبارة الأخيرة. استخدام هياكل التحكم يمكننا من تغيير تنفيذ

التعليمات والقفز فوق بعض التعليمات بدون تنفيذ. ويدعم Basic في Crystal Reports معظم هياكل التحكم التى يستخدمها Visual Basic. تشمل هذه الهياكل:

If Statements
Select Statements
For/Next Loops
Do Loops
While Loops
Preventing Infinite Loops

التفاعل مع المستخدمين

يتطلب التحكم فى التقارير أثناء التشغيل إضافة كود إلى التطبيق يدعم إدخالات المستخدم، عن طريق استخدام كائنات مولد التقارير أو كائن مشاهدة تقارير الويندوز. ويمكن تدعيم هذه الإدخالات فى تطبيق Crystal Reports باستخدام الطرق التالية:

- تكوين إجراءات معالجة الأحداث التى تقع أثناء تعامل المستخدم مع متحكم مشاهدة نماذج الويندوز (CrystalReportViewer).
- تكوين إجراءات معالجة الأحداث التى تقع أثناء قيام المستخدم بالتعامل مع أدوات التحكم الأخرى على نموذج الويندوز.

تعديل متحكم مشاهدة التقارير

يدعم تصنيف متحكم مشاهدة نماذج الويندوز خيارات التحكم فى طريقة عرض هذه الأداة وقت التشغيل. وتشمل عمليات التحكم، إظهار وحجب شريط الأدوات، إظهار وحجب شجرة المجموعات.

إظهار وحجب شريط الأدوات

يمكن السماح للمستخدم بالتحكم فى عرض وإخفاء شريط الأدوات (Toolbar) فى متحكم مشاهدة التقارير بتطبيقات الويندوز باستخدام خاصية DisplayToolbar فى تصنيف CrystalReportViewer باستخدام الكود التالى :

```
crystalReportViewer1.DisplayToolBar = True
```

حجب وإظهار شجرة المجموعات (Group Tree)

يمكن السماح للمستخدم بحجب وإخفاء شجرة المجموعات (Group Tree) فى متحكم مشاهدة التقارير بتطبيقات الويندوز باستخدام خاصية DisplayGroupTree التابعة لتصنيف CrystalReportViewer، كما يتضح من الكود التالى :

```
CrystalReportViewer1.DisplayGroupTree = True
```

تغيير التقرير المطلوب مشاهدته

يمكننا السماح للمستخدم بتحديد التقرير الذى يريد مشاهدته فى وقت التشغيل عن طريق استخدام متحكم Viewer. لتنفيذ ذلك، نستخدم خاصية ReportSource فى تصنيف Crystal Report viewer. يمكن ضبط هذه الخاصية على اسم أحد الملفات، اسم متغير كائن تقرير، أو اسم تقرير نوعى. توضح الخطوات التالية كيفية قيام المستخدم باختيار تقرير من مربع حوار Open File، ثم ربط التقرير مع متحكم Viewer لكى يستطيع المستخدم مشاهدة التقرير فى وقت التشغيل.

لاختيار تقرير فى وقت التشغيل:

١. نبدأ مشروع جديد.

٢. نضيف متحكم Crystal Report Viewer إلى النموذج.

٣. نضيف متحكم Button إلى النموذج.

٤. نضيف متحكم OpenFileDialog إلى النموذج.

٥. ننقر نقرا مزدوجا بالماوس على متحكم Button لتكوين إجراء معالجة حدث Click ثم ندرج الكود التالى فى إجراء المعالجة :

```
openFileDialog1.Filter = "Crystal Reports|*.RPT"
If openFileDialog1.ShowDialog () = DialogResult.OK Then
  crystalReportViewer1.ReportSource = openFileDialog1.FileName
End If
```

تعديل أطقم الحروف والألوان

يسمح لنا مولد Crystal Report بإضافة لون وأطقم حروف خاصة إلى الحقول على التقرير. يتيح لنا ذلك من التركيز على البيانات الهامة وتكوين تقارير محترفة. ويمكن

للمستخدم تعديل التقرير فى وقت التشغيل بتغيير هذه الضوابط. لتغيير ألوان بيانات التقرير فى وقت التشغيل :

١. نكون تقرير ونضيف صيغة إلى مقدمة التقرير.

٢. نربط التقرير مع متحكم Viewer.

٣. نضيف متحكم Button إلى النموذج. حيث يستخدم هذا الزر فى تغيير لون البيانات.

٤. ننقر نقرا مزدوجا على متحكم Button لإدخال كود فى إجراء معالجة حدث Click

```
Dim report As ReportDocument = New ReportDocument ()
Dim section As Section
Dim fieldObject As FieldObject
Dim fieldFormat As FieldFormat
report.Load ("d:\vbProjects\windowsApplication2\CrystalReport1.rpt")
section = report.ReportDefinition.Sections.Item("Section3")
If section.ReportObjects ("Field1").Kind = ReportObjectKind.FieldObject Then
   fieldObject = section.ReportObjects ("Field1")
   fieldFormat = fieldObject.FieldFormat
   fieldObject.Color = Color.Red
End If
```

٥. نجدد معلومات التقرير بإضافة الكود التالى :

```
CrystalReportViewer1.ViewReport
```

التحكم فى تقديم البيانات على التقرير

يمكن السماح للمستخدم باختيار البيانات التى تعرض، واختيار كيفية عرضها باستخدام المعاملات أو صيغ الاختيار لتكوين المجموعات وفرز البيانات.

التحكم فى حقول المعاملات وقت التشغيل

يمكن تدعيم إدخالات المستخدم باستخدام المعاملات فى Crystal Reports. وتستخدم المعاملات لأغراض متنوعة. من أمثلة هذه الأغراض ما يلى:

- بناء المعامل على أساس حقل قاعدة بيانات ثم تمكين المستخدم من تحديد قيم هذه الحقل التى يتم على أساسها فرز البيانات بالتقرير.

- استخدام حقول المعاملات فى تطبيق التنسيق المشروط على التقرير.
- ترتيب الفرز باستخدام حقول المعاملات.

يوضح المثال التالى كيفية ضبط قيم حقل المعامل، باستخدام الكود، فى وقت التشغيل. ويشرح كيفية ضبط معاملين مختلفين. المعامل الأول يحتوى على قيم فردية متعددة والمعامل الثانى يحتوى على نطاق من البيانات:

```
Dim paramFields As New ParameterFields ()
Dim paramField As New ParameterField ()
Dim discreteVal As New ParameterDiscreteValue ()
Dim rangeVal As New.ParameterRangeValue ()

paramField.ParameterFieldName = "Customer Name"
discreteVal.Value = "AIC Childrens"
paramField.CurrentValues.Add (discreteVal)

discreteVal = New ParameterDiscreteValue()
discreteVal.Value = "Aruba Sport"
paramField.CurrentValues.Add(discreteVal)

paramFields.Add (paramField)

paramField = New ParameterField()

paramField.ParameterFieldName = "Customer ID"

rangeVal.StartValue = 42
rangeVal.EndValue = 72
paramField.CurrentValues.Add (rangeVal)

paramFields.Add (paramField)

crystalReportViewer1.ParameterFieldInfo = paramFields

crystalReportViewer1.ReportSource = "c:\reports\my report.rpt"
```

تعديل صيغ الاختيار وقت التشغيل

يمكن استخدام صيغ الاختيار لفرز السجلات التى نريد أن يشتمل عليها تقرير. ويمكن

أيضا تحديد حقول للاستخدام فى توزيع البيانات على مجموعات وفرز هذه البيانات. المثال التالى يمكن المستخدم من تعديل صيغة الاختيار وقت التشغيل لتقرير. ويمكن أن يتم ذلك من خلال متحكم Viewer أو من خلال محرك التقرير.

لتعديل البيانات فى وقت التشغيل :

١. نكون تقريرا يستخدم جدول العملاء فى قاعدة البيانات العينة، xtreme.mdb.

٢. نضيف صيغة اختيار تحتوى على الكود التالى:

```
{Customer.Last Year's Sales} > 11000.00
```

يترتب على تنفيذ هذه الصيغة إعادة سجلات العملاء الذين بلغت المبيعات عليهم فى السنة السابقة أكثر من ١١٠٠٠ دولار.

٣. نضيف متحكم Viewer إلى النموذج.

٤. نربط التقرير بمتحكم Viewer على النموذج.

٥. نضيف متحكم Text Box ومتحكم Button إلى النموذج.

٦. يقوم المستخدم بإدخال الحد الأدنى للقيمة التى عرضها بالنسبة لمبيعات السنة السابقة فى مربع النص (Text Box) ثم ينقر على الزر (Button) لإدخال ذلك إلى النظام.

٧. ننقر نقرا مزدوجا على الزر السابق لإدخال الكود الخاص بمعالج حدث Click.

٨. نمرر صيغة الاختيار إلى متحكم مشاهدة التقارير باستخدام الكود التالى:

```
Dim SelectFormula As String
SelectFormula = "{Customer.Last Year's Sales} > " & textBox1 ().Text
crystalReportViewer1.SelectionFormula = SelectFormula
```

أو نمرر صيغة الاختيار من خلال كائن التقرير، كما يتضح من الكود التالى:

```
Dim selectFormula As String
selectFormula = "{Customer.Last Year's Sales} > " & textBox1 ().Text
Report.DataDefinition.RecordSelectionFormula = selectFormula
```

٩. نجدد بيانات التقرير عن طريق وضع الكود التالى فى نهاية القسم :

```
CrystalReportViewer1.RefreshReport ()
```

تعديل حقول المجموعات فى وقت التشغيل

يمكن استخدام حقول المجموعات لتقسيم بيانات التقرير إلى مجموعات على أساس الشروط الموضوعة. المثال التالى يمكن المستخدم من تعديل حقل المجموعة الخاص بالتقرير، ويفترض وجود تقرير Crystal به العناصر التالية:

- حقول Customer.City، Customer.Country، و Customer.Region فى قسم التفصيلات.
- ويستخدم التطبيق متحكم ComboBox لتحديد الاختيارات الخاصة بالمجموعة.
- يمكن تغيير مجموعات فى التقرير عن طريق إضافة كود إلى متحكم Button.

لتكوين مجموعات وقت التشغيل:

١. نكون تقرير يستخدم جدول Customer فى قاعدة بيانات xtreme.mdb.

٢. نضيف مجموعة إلى التقرير باستخدام حقل Customer.City.

٣. نربط التقرير بمتحكم Viewer.

٤. نضيف متحكم ComboBox إلى النموذج.

٥. يحتوى متحكم ComboBox على حقول Customer.City، Customer.Country، و Customer.Region لكى تمثل الخيارات المتاحة.

٦. نضيف متحكم Button إلى النموذج لكى يقوم المستخدم بالنقر على هذا الزر لتكوين مجموعات على أساس الحقل الذى تم اختيارة فى متحكم ComboBox.

٧. ننقر نقرا مزدوجا على متحكم Button لإدخال الكود فى إجراء معالجة حدث Click.

```
Dim FieldDef As FieldDefinition
FieldDef = Report.Database.Tables.Item (0).Fields.Item (comboBox1 ().Text)
Report.DataDefinition.Groups.Item (0).ConditionField = FieldDef
```

٨. ندخل الكود التالى فى نهاية قسم الكود لتجديد بيانات التقرير عند التشغيل :

```
CrystalReportViewer1.RefreshReport ()
```

تعديل حقول الفرز فى وقع التشغيل

يمكننا استخدام حقول الفرز لاختيار الحقل الذى على أساسه يجرى فرز البيانات التى فى التقرير. ويمكن أن يكون اتجاه الفرز إلى أعلى، إلى أسفل، الحصول على أعلى القيم، الحصول على أقل القيم. يمكن المثال التالى المستخدم من تعديل حقل الفرز الخاص بأحد التقارير ويفترض أن لدينا تقرير Crystal تتوفر به الشروط التالية :

- يحتوى التقرير على حقلين فى قسم Details : Customer.Name و Customer.Country.
- البيانات يجرى فرزها على أساس حقل Customer.Name.
- يمكن تغيير ترتيب الفرز فى التقرير عن طريق إضافة كود إلى متحكم Button.

لفرز الحقول وقت التشغيل:

١. نكون تقرير يستخدم جدول Customer قاعدة بيانات extreme.mdb.

٢. ننقر بزر الماوس الأيمن على الحقل الذى نريد ضبطة ليكون حقل الفرز الإفتراضى، ننقر Sort Records فى القائمة المختصرة ونحدد هذا الحقل على أنة الحقل الذى يتم الفرز على أساسه، وهو فى هذا المثال حقل Customer.Name.

٣. نربط التقرير مع متحكم Viewer.

٤. نضيف متحكم Button إلى النموذج لكى يستطيع المستخدم النقر علية بغرض تغيير حقل الفرز.

٥. نقوم بالنقر المزدوج على متحكم Button لإدخال الكود فى إجراء معالجة حدث Click.

```
Dim FieldDef As FieldDefinition
FieldDef = _
Report.Database.Tables.Item (0).Fields.Item (comboBox1 ().Text)
Report.DataDefinition.SortFields.Item (0).Field = FieldDef
```

٦. ندخل الكود التالى فى نهاية قسم الكود لتجديد بيانات التقرير عند التشغيل :

```
CrystalReportViewer1.RefreshReport ()
```

تعديل خيارات التصدير

يسمح محرك Crystal Report لنا بتسليم التقارير بطرق مختلفة باستخدام خيار Export . يمكننا إرسال التقرير إلى ملف قرص، إلى مجلد تبادل (Exchange Folder)، أو إلى بريد إلكتروني. ويمكن تصدير التقرير إلى Excel، Crystal Reports، Microsoft Word، HTML، RTF. وهناك ثلاثة طرق لضبط خيارات المستخدم الخاصة بالتصدير :

- استخدام زر Export Expert على شريط أدوات CrystalReportViewer.
- تكوين زر خاص يقوم باستدعاء مربع حوار Export Expert. تسمح لنا هذه الوسيلة بإخفاء شريط الأدوات عندما لا نكون فى حاجة إلى وظائفه الأخرى أو عندما نريد تعديل واجهة الاستخدام.
- نكتب كود لتحديد نوع خاص من التصدير والمقصد.

تصدير التقارير باستخدام زر *Export*

يقع زر Export فى الجانب الأعلى على اليسار بشريط أدوات Crystal Report Viewer. وعندما يقوم المستخدم بالنقر على هذا الزر، يتم عرض مربع حوار لتحديد طريقة التصدير. ويظهر زر Export أتوماتيكيا إلا إذا قمنا بحجب شريط أدوات متحكم Viewer. وفى جميع الأحوال، يمكن حث المستخدم على إدخال خيارات التصدير، أو نستخدم الكود للتصدير باستخدام خيارات محددة. ويمكن تكوين زر Export خاص بنا بدلا من الزر الإفتراضى الموجود على شريط Viewer. لتكوين زر تصدير خاص بنا، نطبق الخطوات التالية:

١. نضيف متحكم Button إلى النموذج.

٢. نقوم بالنقر المزدوج على الزر (Button) لإدخال الكود التالى فى إجراء معالجة حدث Click:

```
crystalReportViewer1.ExportReport ()
```

٣. عندما يقوم المستخدم بالنقر على الزر ، سوف يظهر مربع حوار Export ليحث المستخدم على اختيار أحد أشكال التصدير المختلفة التى يقدمها Crystal Reports.

ضبط خيار التصدير باستخدام الكود

يمكن ضبط خيارات تصدير التقرير عن طريق ضبط مقصد التصدير، الشكل، واسم الملف داخل الكود. ويجب ملاحظة أنة لا توجد قيم افتراضية لحقول DestinationType، FormatType، أو DiskFileName؛ ولهذا، يجب تحديد قيم لكل الحقول الثلاثة. يمكن أن يتم ذلك من خلال محرك التقرير (Report Engine). المثال التالى يقوم بتصدير التقرير إلى قرص باستخدام نموذج Excel 8.0:

١. نضيف متحكم Button إلى النموذج.

٢. فيما يتعلق بخاصية Text فى الزر السابق إضافته فى الخطوة السابقة ، نطبع Export to MS Excel.

٣. نقوم بالنقر المزدوج على الزر وندخل الكود التالى فى إجراء معالجة حدث Click :

```
Dim exportOpts As New ExportOptions ()
Dim diskOpts As New DiskFileDestinationOptions ()
Dim excelFormatOpts As New ExcelFormatOptions ()
exportOpts = Report.ExportOptions
excelFormatOpts.ExcelTabHasColumnHeadings = true
exportOpts.ExportFormatType = ExportFormatType.Excel
exportOpts.FormatOptions = excelFormatOpts

exportOpts.ExportFormatType = ExportFormatType.Excel

exportOpts.ExportDestinationType = ExportDestinationType.DiskFile

diskOpts.DiskFileName = fileName
exportOpts.DestinationOptions = diskOpts

Report.Export ()
```

الوصول إلى قواعد البيانات الآمنة

فى تطبيقات الويندوز التى تحتوى على تقارير Crystal، يقوم مربع حوار تلقائيا بحث المستخدمين على إدخال اسم المستخدم وكلمة المرور. ويمكن استخدام الكود لتحديد اسم المستخدم وكلمة المرور. يوضح المثال التالى كيفية تمرير معاملات بدء تسجيل الدخول على جداول التقرير الموجودة فى قاعدة بيانات SQL Server:

١. نبدأ مشروع جديد.

٢. نضيف متحكم زر (Button) و أربعة مربعات نص (TextBox) إلى النموذج.

٣. نخصص أسماء ServerNameTxt، dbNameTxt، usserNameTxt، و passwordTxt لمربعات النص الأربعة. .

٤. نقوم بالنقر المزدوج على متحكم Button لإدخال الكود التالى الخاص بإجراء معالجة حدث Click.

```
Dim logOnInfo As New TableLogOnInfo ()
Dim i As Integer

For i = 0 To report.Database.Tables.Count - 1
  logOnInfo.ConnectionInfo.ServerName = serverNameTxt.Text
  logOnInfo.ConnectionInfo.DatabaseName = dbNameTxt.Text
  logOnInfo.ConnectionInfo.UserID = userNameTxt.Text
  logOnInfo.ConnectionInfo.Password = passwordTxt.Text
  report.Database.Tables.Item (i).ApplyLogOnInfo (logOnInfo)
Next i
```

وتستخدم نفس الوسيلة بالنسبة لقواعد البيانات الشخصية، مثل Microsoft Access و Paradox. ويجب عدم إدخال قيم فى المتغيرات LogOnInfo.ServerName و LogOnInfo.DatabaseName.

التعامل مع الإستثناءات

عند تشغيل Crystal Reports من خلال تطبيقاتنا، قد تكون هناك حالات يسبب فيها محرك Crystal Reports أخطاء استثنائية. وترجع هذه الأخطاء إلى العديد من الأسباب، التى نعرض بعضها فيما يلى:

- لا يستطيع محرك التقرير أن يتصل مع قاعدة البيانات. يمكن أن يحدث ذلك بسبب إدخال معاملات خاطئة لبدء التشغيل. ويمكن أن يحدث أيضا بسبب أخطاء أخرى بقاعدة البيانات، مثل إقفال جدول بيانات بواسطة مستخدم أخر، تم تثبيت محرك قاعدة البيانات بطريقة غير ملائمة، أو أن الجدول قد أصبح غير صالح للاستخدام.

- تمريـر بـيانات غير صحيحة إلى المعاملات أثناء التشغيل، مثل تمرير نص إلى حقل معامل رقمي.
- وجود خطأ فى إحدى صيغ Crystal Reports. قد يرجع ذلك إلى خطأ لغوى فى الصيغة أو أخطاء أخرى، مثل القسمة على صفر.
- فشل محرك التقارير فى فتح أحد التقارير. يمكن أن يحدث ذلك عندما يكون اسم الملف أو المسار غير صحيح، أو أن التقرير ليس من بين تقارير Crystal Reports.

يمكن أن ندع محرك Crystal Reports يتعامل مع الاستثناء بنفسه، ويعرض رسالة الخطأ الخاصة به، ويمكن أن نختار معالجة الاستثناء بأنفسنا.

التعامل مع الإستثناءات باستخدام الكود

يمكن استخدام كائن Report Document أو استخدام كائن Viewer للتعامل مع الأخطاء الاستثنائية باستخدام الكود. عند معالجة خطأ استثنائي باستخدام متحكم Viewer ، يمكننا استخدام حدث HandleException. المثال التالى يوضح كيفية الهيمنة على رسالة الخطأ التى تعرض عندما يكون هناك مشكلة خاصة بتسجيل بدء تشغيل قاعدة بيانات أو الوصول إلى مصدر بيانات. وكل رسائل الأخطاء الأخرى سوف تترك لكى يقوم محرك Crystal Reports بعرضها.

```
Private Sub CrystalReportViewer1_HandleException (ByVal source As Object, ByVal
e As CrystalDecisions.Windows.Forms.ExceptionEventArgs) Handles
CrystalReportViewer1.HandleException
  If TypeOf (e.Exception) Is EngineException Then
    Dim engEx As EngineException
    engEx = e.Exception
    If engEx.ErrorID = EngineExceptionErrorID.DataSourceError Then
      e.Handled = True
      MessageBox.Show _
      ("An error has occurred while connecting to the database.")
    ElseIf engEx.ErrorID = EngineExceptionErrorID.LogOnFailed Then
      e.Handled = True
      MessageBox.Show _
   ("Incorrect Logon Parameters. Check your user name and password.")
    End If
  End If
```

```
End Sub
```

وعند معالجة الأخطاء الاستثنائية باستخدام كائن ReportDocument، نحتاج إلى استخدام عبارة try-catch. في فقرة catch، يمكن استخدام واحدا من تصنيفات Crystal Exception. تصنيف EngineException هو التصنيف الأساسي الذي سوف يقوم باصطياد الأخطاء الاستثنائية التي يطلقها محرك Crystal Reports. ويمكن استخدام التصنيفات الأخرى لتحديد نوع الخطأ الذي حدث.

المثال التالي يوضح كيفية الهيمنة على رسالة الخطأ التي تعرض عند حدوث مشكلة متعلقة بتسجيل بدء تشغيل قاعدة بيانات أو الوصول إلى مصدر بيانات. وكل الإستثناءات الأخرى سوف تعرض رسالة الخطأ الأصلية.

```
Private Sub Button1_Click (ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles Button1.Click
  Try
    Dim report As New ReportDocument ()
    report.Load ("c:\sample.rpt")
    report.PrintToPrinter (1, True, 1, 2)
  Catch engEx As LogOnException
    MessageBox.Show _
   ("Incorrect Logon Parameters. Check your user name and password.")
  Catch engEx As DataSourceException
    MessageBox.Show
    ("An error has occurred while connecting to the database.")
  Catch engEx As EngineException
    MessageBox.Show (engEx.Message)
  End Try
End Sub
```

المحتويات

الفصل الثانى : مكونات لغة Visual Basic.NET

الفصل الثالث : تطبيقات الويندوز

الفصل الرابع : تكوين واجهات استخدام التطبيقات

الفصل الخامس : قواعد البيانات

الفصل السادس : التقارير